CVE-2022-36436: Twisted VNC Authentication Proxy authentication bypass

No Comments 1

By Marios Levogiannis | August 30, 2022 | Blog

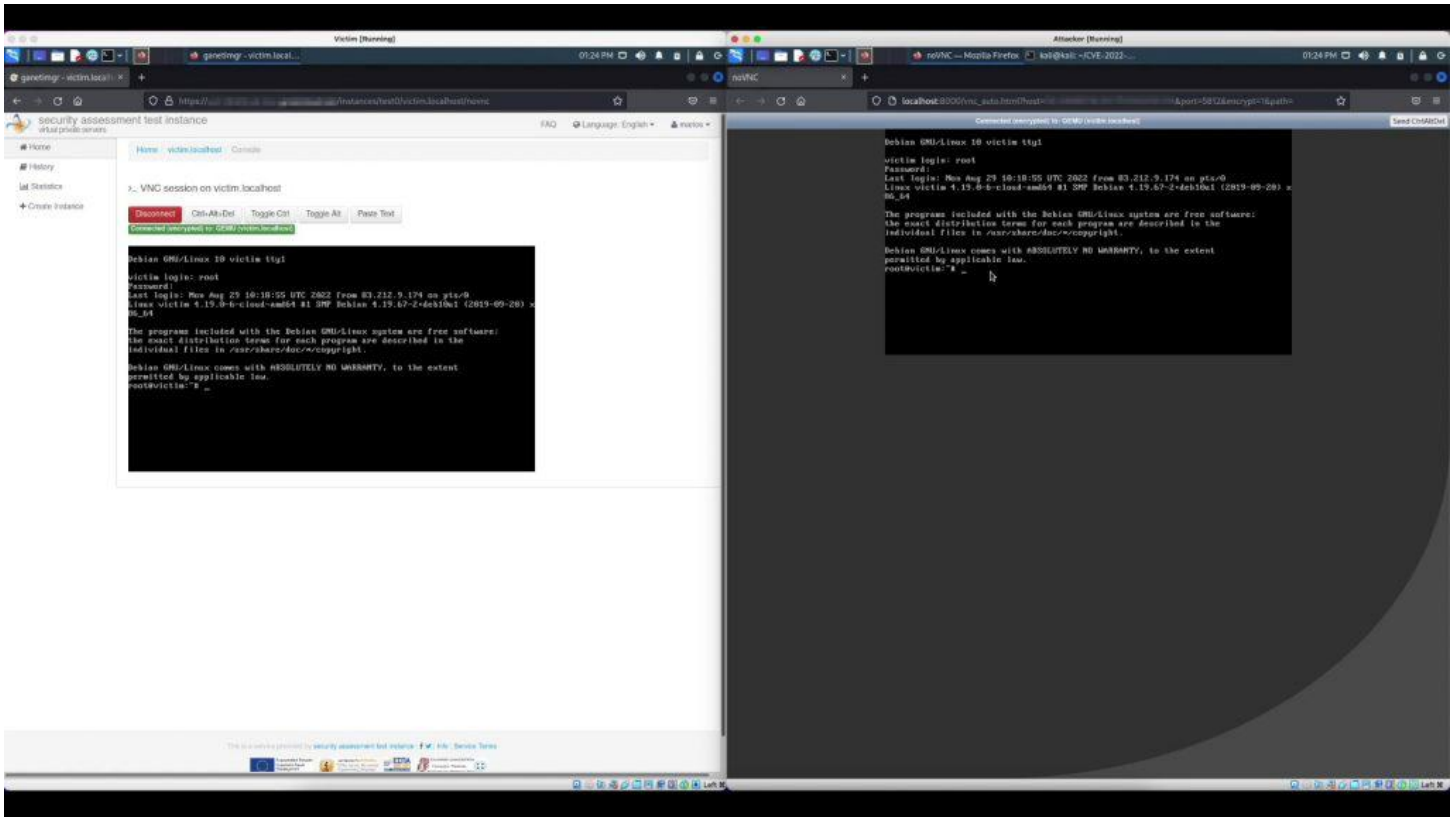


Table of Contents

Introduction

GRNET provides virtualization services on top of a custom virtualization infrastructure. In this context, we offer to our users optional VNC (Virtual Network Computing) access to their VMs. On the backend, the infrastructure utilizes OSU Open Source Lab's Twisted VNC Authentication Proxy (also known as VNCAuthProxy) to dynamically allow multiple clients to connect to a VM's VNC console using different passwords.

During a periodic security audit of our infrastructure we discovered a new security vulnerability in the Twisted VNC Authentication Proxy. The vulnerability allows an attacker to bypass the proxy server's authentication mechanism and connect to a VNC server for which the proxy server is accepting connections. Exploiting this vulnerability, the attacker can gain access to a shared VNC console with the victim, effectively giving the attacker the same level of access to the the target VM as the victim. The vulnerability affects all versions up to version 1.1.1 (commit a399697). Further, a CVE entry was assigned to the vulnerability: CVE-2022-36436.

After discovering the vulnerability we followed a responsible disclosure approach, i.e. contacted the vendor and worked together to create a fix. The vulnerability was fixed in version 1.2.0 (commit edc149a). Since the new version has been released, we now publicly disclose the details of the vulnerability and how to identify if it has been exploited.

According to our analysis the vulnerability had not been exploited in our infrastructure.

Vulnerability details

The Twisted VNC Authentication Proxy is a proxy server for VNC with man-in-the-middle authentication, which can be used as a gateway to VNC servers behind protected networks. It implements only the handshake messages of the RFB ("remote framebuffer") protocol (`ProtocolVersion`, `Security`, `SecurityResult`). All other messages of the RFB protocol are simply proxied between the client and the server.

We identified that the proxy server does not enforce the "VNC Authentication" security type (password authentication) during the handshake, allowing the client to select the "None" security type (no authentication) and connect to the VNC server without providing any authentication credentials.

Specifically, during handshake the proxy server sends to the client the security types it supports. The message with the supported security types is hardcoded in the code of the proxy server (file `vncap/vnc/protocol.py`, line 77). The first byte of the message is the number of the security types supported and the following bytes are the security type numbers. In the following case, the message is `\x02\x01\x02`, which decodes to a list of two supported security types, 1 ("None") and 2 ("VNC Authentication").

```
File: vncap/vnc/protocol.py
68:     def check_version(self, version):
69:         """
70:         Determine the client's version and decide whether to continue the
71:         handshake.
72:         """
73:
74:         if version == self.VERSION:
75:             log.msg("Client version %s is valid" % version.strip())
76:             # Hardcoded: 2 security types: None and VNC Auth.
77:             self.transport.write("\x02\x01\x02")
78:             return self.select_security_type, 1
79:         else:
80:             log.err("Can't handle VNC version %r" % version)
81:             self.transport.loseConnection()
```

Then, the client responds with one byte, which is the number of the security type of their choice. The proxy server always accepts whatever the client has selected (file `vncap/vnc/protocol.py`, lines 88, 90, 96).

```
File: vncap/vnc/protocol.py
083:     def select_security_type(self, security_type):
084:         """
085:         Choose the security type that the client wants.
086:         """
087:
088:         security_type = ord(security_type)
089:
090:         if security_type == 2:
091:             # VNC authentication. Issue our challenge.
092:             self.challenge = urandom(16)
093:             self.transport.write(self.challenge)
094:
095:             return self.vnc_authentication_result, 16
096:         elif security_type == 1:
097:             # No authentication. Just move to the SecurityResult.
098:             self.authenticated()
099:         else:
100:             log.err("Couldn't agree on an authentication scheme!")
101:             self.transport.loseConnection()
```

From the example above we can deduce that if the client responds with the “None” security type, the proxy server will mark the client as authenticated and skip the authentication flow (file `vncap/vnc/protocol.py`, lines 96-98).

Impact

The attacker can either gain access to the victim’s VNC session or force the victim to disconnect from the session. This is controlled by the attacker using the `shared-flag` option in the `ClientInit` message that is sent

from the VNC client to the VNC server after the handshake messages have been exchanged with the proxy server.

By setting the `shared-flag` option to `true`, the attacker can gain access to the same VNC session with the victim, effectively giving the attacker the same level of access to the the target system as the victim. As a result, the attacker may let the victim authenticate to the system with their credentials first and then run arbitrary commands in the victim's authenticated shell using the shared VNC session.

Alternatively, the attacker can set the `shared-flag` option to `false` which instructs the VNC server to disconnect the victim's VNC client, effectively allowing the attacker to make the VNC service unavailable.

Exploitation Requirements

Available time frame

The attacker cannot connect to arbitrary upstream servers on demand, but only to upstream servers for which the proxy server is currently accepting connections, i.e. for 30 seconds after the respective command has been received on the control port (file `vncap/control.py`, line 55).

```
File: vncap/control.py
20: class ControlProtocol(LineReceiver):
21:
22:     def lineReceived(self, line):
23:         log.msg("Received line %s" % line)
24:         try:
...
50:             # Set up our timeout.
51:             def timeout():
52:                 log.msg("Timed out connection on port %d" % sport)
53:                 listening.stopListening()
54:                 self.factory.free_port(sport)
55:             reactor.callLater(30, timeout)
```

Optional client IP verification

The proxy server can be configured via the control port to perform a verification of the client's IP address. The exploitation of this vulnerability is only possible if the IP address verification is disabled (the default behavior) or if both the victim and the attacker share the same IP address (e.g. they are behind the same NAT) (file `vncap/vnc/protocol.py`, lines 113-118).

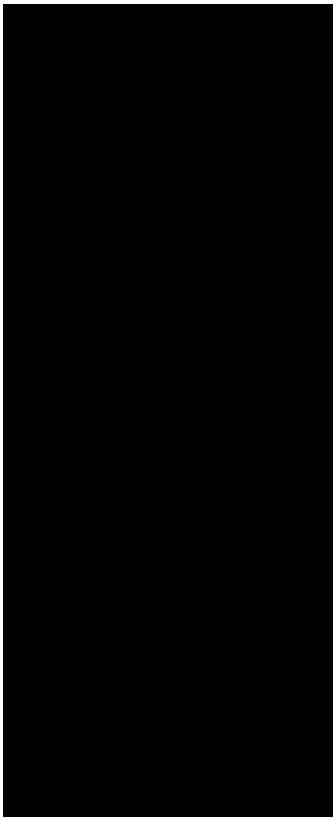
```
File: vncap/vnc/protocol.py
45: class VNCServerAuthenticator(VNCAuthenticator):
...
112:     def verify_ip(self):
113:         if 'ip' in self.options:
114:             if self.options['ip'] != self.transport.getPeer().host:
```

```
115:             log.err("Failed to verify client IP")
116:             self.transport.loseConnection()
117:         else:
118:             log.msg("Verified client IP")
```

Exploitation

When the proxy server is notified via the control port to proxy new connections to a VNC server, it opens a port in the range 5800-5899 and listens for new connections for 30 seconds. An attacker can repeatedly scan ports 5800-5899 of the target instance of the proxy server and when a port is found to be open (i.e. the proxy server has started to accept connections) they can connect using a VNC client and select the “None” security type to bypass the authentication flow.

The following video demonstrates the exploitation of the vulnerability in a vulnerable instance of the Twisted VNC Authentication Proxy. The left and right halves of the screen show the victim’s and the attacker’s computer, respectively. The attack begins with the attacker starting a script to scan the ports of the target instance of the proxy server. Then, the victim uses the administration frontend for the VMs to open a new VNC console to one of their VMs. When the attacker’s script detects the open port, it starts a noVNC client (modified to always select the “None” security type) to open a new VNC console in shared mode. Finally, the victim authenticates to the system using their credentials, and since the VNC session is shared between the two consoles, the attacker gets unrestricted access to the victim’s authenticated shell.



0:00 / 0:21

Indicators of compromise

The proxy server logs a message on every handshake message of the Security handshake of the RFB protoc
If the “VNC Authentication” security type is selected by the client, the message “Doing VNC auth, buf ” is logged, as shown in the code snippet below (lines 90-95, 104). If the “None” security type is selected by the client, then the aforementioned message is skipped (lines 96-98).

```
File: vncap/vnc/protocol.py
045: class VNCServerAuthenticator(VNCAuthenticator):
...
083:     def select_security_type(self, security_type):
...
090:         if security_type == 2:
091:             # VNC authentication. Issue our challenge.
092:             self.challenge = urandom(16)
093:             self.transport.write(self.challenge)
094:
095:             return self.vnc_authentication_result, 16
096:         elif security_type == 1:
097:             # No authentication. Just move to the SecurityResult.
098:             self.authenticated()
099:         else:
100:             log.err("Couldn't agree on an authentication scheme!")
101:             self.transport.loseConnection()
102:
103:     def vnc_authentication_result(self, response):
104:         log.msg("Doing VNC auth, buf %r" % response)
105:
106:         if check_password(self.challenge, response, self.password):
107:             self.authenticated()
108:         else:
109:             log.err("Failed VNC auth!")
110:             self.transport.loseConnection()
```

The following excerpt is from a legitimate authentication flow. It can be seen that the aforementioned message is included.

```
[VNCServerAuthenticator (WebSocketProtocol) (TLSMemoryBIOProtocol),0,XXX.XXX.XXX.XXX] Client version RFB 003.008 is valid
[VNCServerAuthenticator (WebSocketProtocol) (TLSMemoryBIOProtocol),0,XXX.XXX.XXX.XXX] Doing VNC auth, buf '\x05\xbe\t\xd6\xfb\xea\xc8\xbbs6\xca.0c\xcf\xab'
[VNCServerAuthenticator (WebSocketProtocol) (TLSMemoryBIOProtocol),0,XXX.XXX.XXX.XXX] Successfully authenticated a client!
[VNCServerAuthenticator (WebSocketProtocol) (TLSMemoryBIOProtocol),0,XXX.XXX.XXX.XXX] Successfully authenticated <vncap.vnc.protocol.VNCServerAuthenticator instance at 0x7fbbecc604d0>!
```

The following excerpt is from an unauthorized authentication flow. It can be seen that the aforementioned message is missing.

```
[VNCServerAuthenticator (WebSocketProtocol) (TLSMemoryBIOProtocol),2,XXX.XXX.XXX.XXX] Client version RFB 003.008 is valid
```

```
[VNCServerAuthenticator (WebSocketProtocol) (TLSMemoryBIOProtocol),2,XXX.XXX.XXX.XXX] Successfully authenticated a client!
[VNCServerAuthenticator (WebSocketProtocol) (TLSMemoryBIOProtocol),2,XXX.XXX.XXX.XXX] Successfully authenticated <vncap.vnc.protocol.VNCServerAuthenticator instance at 0x7fbbecb83d88>!
```

Apart from the actual message, each log line also contains the client's IP (`xxx.xxx.xxx.xxx` in the examples) and the connection number for the current listener (`0` and `2` in the example, respectively).

To detect whether the vulnerability has been exploited in the past or not, one must count the "Doing VNC auth, buf <hex-data>" and "Successfully authenticated a client!" messages for each client IP address and connection number pair. If the number of the "Successfully authenticated a client!" messages is greater than the number of the "Doing VNC auth, buf <hex-data>" messages, an exploitation took place.

We have implemented the following script to automate the process above:

▶ Show code

Recommendation

It is recommended to update all instances of the Twisted VNC Authentication Proxy to version 1.2.0 (or above), which includes the fix for this authentication bypass vulnerability.

Conclusion

The Twisted VNC Authentication Proxy authentication bypass is a critical vulnerability that system administrators must be aware of. It is also an interesting use case that highlights how implementation errors may lead to privilege escalation.

As part of the response to the discovery, we checked our Twisted VNC Authentication Proxy's logs to determine if there was an exploitation in the GRNET infrastructure. After assessing the logs (gathered continuously for 2 years) we concluded that our infrastructure was intact.

We would like to thank the OSU Open Source Lab for their timely response and cooperation in handling this security vulnerability.

Vulnerability timeline

July 5, 2022: Vulnerability identified.
July, 11, 2022: Temporary fix applied to GRNET's internal systems.
July 12, 2022: Vendor contacted.
July 25, 2022: CVE ID assigned.

July 29, 2022: Fix released.
August 30, 2022: Public advisory (this document).

References

- https://github.com/osuosl/twisted_vncauthproxy
- https://github.com/osuosl/twisted_vncauthproxy/commit/edc149af29242178091b2d6fcd42c3ef0851644b
- https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-36436

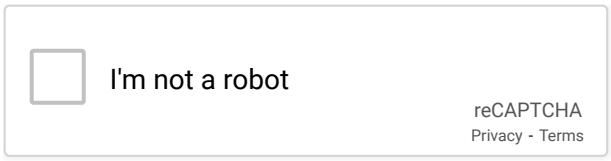
Tags: CVE Twisted VNC Authentication Proxy VNCAuthProxy Vulnerability Research

Leave a Reply

Name *

Email *

Website



Submit Comment

FIND US

7, Kifisias Av. GR 115 23 Athens

Tel: + 30 210 7474274, Fax: +30 210 7474490
Email: info-at-grnet.gr

COOKIE NOTIFICATION

This site uses cookies to deliver our services. If you want to know more about cookies or would like to delete your Cookies, visit our Cookie Policy.

© 2022 GRNET CERT.