cp<r> CHECK POINT RESEARCH BY CHECK POINT

SEARCH

WOULD YOU LIKE SOME RCE WITH YOUR GUACAMOLE?

July 2, 2020

Research by: Eyal Itkin

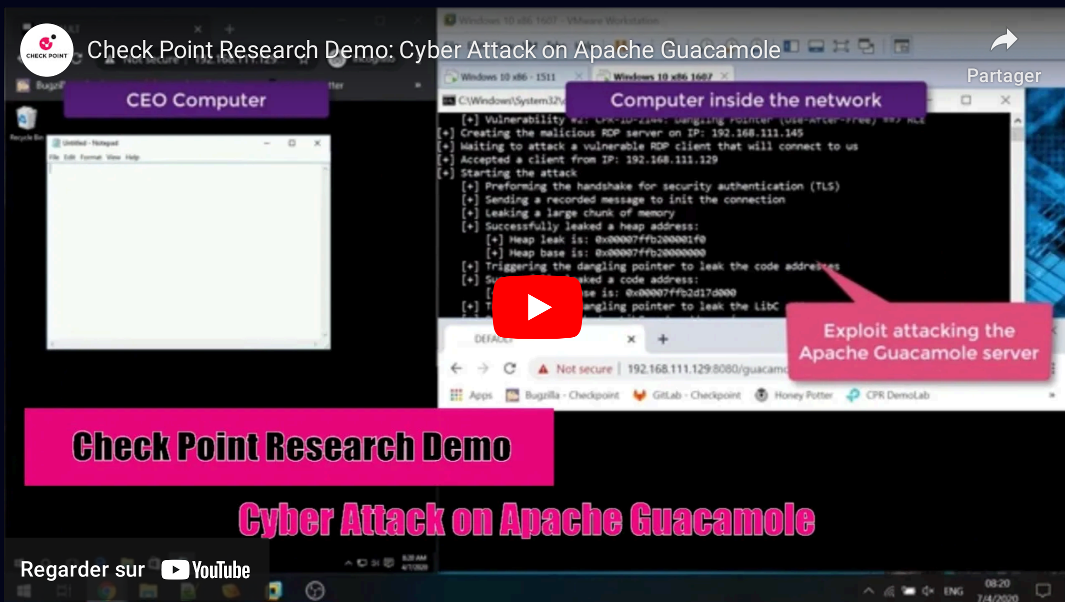
Overview

In many companies, the daily routine involves coming to the office each day to work on your company computer, safely inside the corporate network. Once in a while, a worker may need special offsite access and will connect to the company's network remotely, using one of several available tools.

However, since the outbreak of the COVID-19 pandemic, this daily routine has been reversed. In Check Point, like in many other companies world-wide, the vast majority of work is now done remotely, mostly from home. This transition from onsite to off premise work means that IT solutions for remotely connecting to the corporate network are now used more than ever. This also means that any security vulnerability in these solutions will have a much greater impact, as companies rely on this technology to keep their businesses functioning.

Apache Guacamole is a popular infrastructure for remote work, with more than **10 Million** docker downloads worldwide. In our research, we discovered that Apache Guacamole is vulnerable to several **critical** Reverse RDP Vulnerabilities, and is also impacted by a few new vulnerabilities found in FreeRDP. In short, these vulnerabilities allow an attacker, who has already successfully compromised a computer inside the organization, to launch an attack on the Guacamole gateway when an unsuspecting worker tries to connect to an infected machine. The malicious actor can then achieve full control over the guacamole-server, and intercept and control all other connected sessions.

In this short video demo, we demonstrate how we managed to exploit these vulnerabilities and successfully take control of the Guacamole gateway and all of the connected sessions:



Introducing Apache Guacamole

Intrigued by this shift to mostly working from home, we decided that technological solutions for remote-work would make for an interesting research topic. And indeed, right away, our IT department kindly requested us to review one such solution [Apache Guacamole](https://guacamole.apache.org/). As mentioned previously, it's one of the more prominent tools on the market. Not only that many organizations use this product to connect to their networks, many network accessibility and security products embedded Apache Guacamole inside their own products as well. This list includes: Jumpserver Fortress, Quali, Fortigate, and the list goes on.

After some reconnaissance, we drew a basic sketch of the recommended network architecture, as shown in Figure 1:

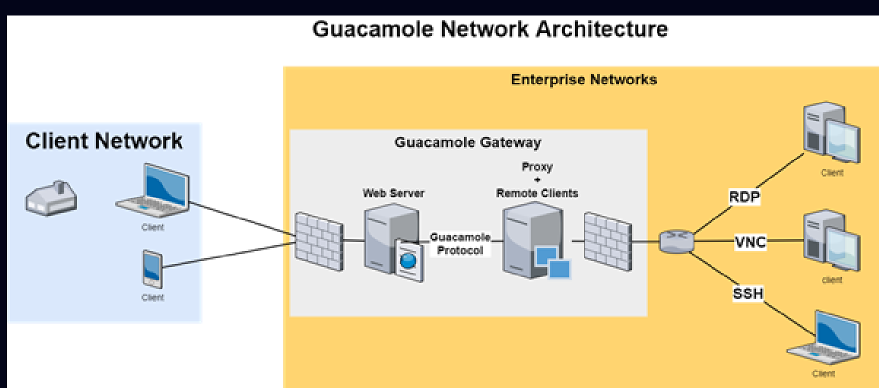


Figure 1: Typical network architecture for deploying the Apache Guacamole gateway.

In essence, an employee uses a browser to connect to his company's internet-facing server, goes through an authentication process, and gets access to his corporate computer. While the employee only uses his browser, the guacamole-server selects one of the supported protocols (RDP, VNC, SSH, etc.) and uses an open source **client** to connect to the specific corporate computer. Once connected, the guacamole-server acts as a middle-man that relays the events back and forth while translating them from the chosen protocol to the special "Guacamole Protocol", and vice versa.

Now that we understand what the architecture looks like, here are a few promising attack vectors to examine:

- **Reverse Attack Scenario:** A compromised machine inside the corporate network leverages the incoming benign connection to attack the gateway, aiming to take it over.
- **Malicious Worker Scenario:** A rogue employee uses a computer inside the network to leverage his hold on both ends of the connection and take control of the gateway.

Do we even need a 0-Day?

Before we dive into the code, let's focus briefly on FreeRDP. In our previous research on the Reverse RDP Attack, we found several critical vulnerabilities in this RDP client which exposed it to attack from a malicious RDP "server." In other words, a malicious corporate computer can take control of an unsuspecting FreeRDP client that connects to it. We even had a basic PoC for one of our vulnerabilities (CVE-2018-8786) and we've demonstrated Remote Code Execution (RCE).

By looking at the released versions of Apache Guacamole, we can see that only version 1.1.0, released at the end of January 2020, added support for the latest FreeRDP version (2.0.0). Knowing that our vulnerabilities in FreeRDP were only patched on version 2.0.0-rc4, this means that **all** versions that were released **before January 2020** are using **vulnerable** versions of FreeRDP.

We could have stopped here and estimated the high probability that most companies haven't yet upgraded to the latest versions, and could already be attacked using these known 1-Days. However, we decided to once again search for vulnerabilities in the RDP protocol, and more specifically:

- The code of guacamole-server, while only focusing on the support for the RDP protocol.
- The code of the latest released version of FreeRDP: version 2.0.0-rc4.

On top of all that, our exploit conditions are that it be able to work on default installations, using only features that are enabled by default, and hopefully not require any interaction from the client. Let's start.

Looking for new vulnerabilities

Being familiar with the code of FreeRDP, and with RDP as a whole, really helped during this security audit. We quickly started finding vulnerabilities.

CPR-ID-2141 – Our first Information Disclosure

CVE: CVE-2020-9497*
File: `protocols\rdp\channels\rdpsnd\rdpsnd-messages.c`
Function: `guac_rdpsnd_formats_handler()`
Side note: As Apache didn't use a 1:1 mapping between our reported vulnerabilities (CPR-IDs) and the CVE-IDs they issued, we will mostly refer to the vulnerabilities by their (more accurate) CPR-IDs.

To relay the messages between the RDP connection and the client, the developers implemented their own extension for the default RDP channels. One such channel is responsible for the audio from the server, hence unsurprisingly called `rdpsnd` (RDP Sound).

However, as is often the case, the integration point between the guacamole-server and FreeRDP proved to be error prone. The incoming messages are wrapped by FreeRDP's `wStream` objects, and the data should be parsed using this object's API. However, as can be seen in Figure 2, the developers forgot to enforce that the incoming stream object must contain a number of bytes that matches what is declared by the packet.

```
/* Remember position in stream */
Stream_GetPointer(input_stream, format_start);

/* Read format */
Stream_Read_UINT16(input_stream, format_tag);
Stream_Read_UINT16(input_stream, channels);
Stream_Read_UINT32(input_stream, rate);
Stream_Seek_UINT32(input_stream);
Stream_Seek_UINT16(input_stream);
Stream_Read_UINT16(input_stream, bps);

/* Skip past extra data */
Stream_Read_UINT16(input_stream, body_size);
// EI-DBG: No checks that input_stream contains "body_size" bytes.
// EI-DBG: This could be used for a 64KB info leak that will go right back to the RDP server.
Stream_Seek(input_stream, body_size);

/* If PCM, accept */
if (format_tag == WAVE_FORMAT_PCM) {

    /* If can fit another format, accept it */
    if (rdpsnd->format_count < GUAC_RDP_MAX_FORMATS) {
        ...
```

Figure 2: Missing input filtering leading to an out of bounds read.

By sending a malicious `rdpsnd` channel message, a malicious RDP server could cause the client to think that the packet contains a huge amount of bytes, which are in fact memory bytes of the client itself. This in turn causes the client to send back a response to the server with these bytes, and grant the RDP server a massive, heartbleed-style, Information Disclosure primitive.

CPR-ID-2142 – Once again, an Information Disclosure

CVE: CVE-2020-9497
File: `protocols\rdp\channels\rdpsnd\rdpsnd.c`
Function: `guac_rdpsnd_process_receive()`

In the same RDP channel, a different message has a similar vulnerability. This time it sends the Out-of-Bounds data to the connected client, instead of back to the RDP server.

Figure 3: Similar Out-of-Bounds read, this time leaking data to the client.

While useful, this leak sends the information to the client, and we hope to build an exploit without the client even knowing that the gateway was attacked.

CPR-ID-2143 – What a surprise, an Information Disclosure

CVE: CVE-2020-9497
File: `protocols\rdp\plugins\guacai\guacai-messages.c`
Function: `guac_rdp_ai_read_format()`

We were intrigued to find an additional channel, `guacai`, responsible for sound messages. This channel is responsible for the "Audio Input", hence the name `guacai`. Although vulnerable to roughly the same vulnerability as the previous channel, this channel is disabled by default.

Figure 4: Yet another Out-of-Bounds read, just like the first one.

At this point of our research, we found 3 major Information Disclosure vulnerabilities, which should be more than enough for bypassing the ASLR (Address Space Layout Randomization). However, we still need a memory corruption vulnerability to complete our exploit chain. Feeling stuck, we went to look at FreeRDP once again, hoping to find vulnerabilities that we might have missed in our previous research.

FreeRDP, our old friend

There weren't a lot of changes made to the RDP client since we last looked at it; the patched version is still the latest one released to date. As is always the case when looking for vulnerabilities, let's first understand a key design "feature" in the `wStream` type used by this client. In Figure 5, we can see the fields of this struct:

Figure 5: The `wStream` object, used to wrap incoming/outgoing packets.

This is a classic example of a simple stream wrapper:

- `buffer` – Pointer to the beginning of the received packet.
- `pointer` – Pointer to the reading head inside the received packet.
- `length` – Size, in bytes, of the incoming packet.

Before a given field is parsed from the input stream, a check should be made to make sure that the stream is big enough to hold it. Such a check can be seen in Figure 6:

Figure 6: Checking for available input, using `Stream_GetRemainingLength()`.

We can't stress enough how important this input check is. Every time a field is parsed or skipped, the pointer field is advanced accordingly. Later on, when the next check is performed, it looks like this:

Figure 7: Calculating the remaining length using the current stream's head.

As soon as the pointer field passes the end of the incoming packet, this calculation will **underflow**, thus returning a **huge unsigned value** that should represent the negative amount of bytes left. In short, miss one check, and the rest will be useless. This interesting design choice makes FreeRDP **highly vulnerable** to Out-of-Bounds read vulnerabilities, as we soon found out.

Before we present these Out-of-Bounds Read vulnerabilities, it is important to note why we care about them. Usually, a read is only useful if it returns the read bytes, in some way or other, to the attacker. Otherwise, the read can only be used as a way for crashing the program when it tries to access unmapped pages in memory. The Apache Guacamole attack scenario is special because we hold both ends of the connection. If, for example, memory bytes are parsed as graphic updates to the screen, these updates will still be sent to the connected client.

In this attack scenario, every Out-of-Bounds Read vulnerability could likely be turned into a weak, but still useful, Information Disclosure.

CPR-ID-2145 & CPR-ID-2146 – Out-of-Bounds Reads in FreeRDP

Remembering the interesting design flaw in the `wStream` object, all we had to do is to look for read operations which aren't backed by sanitation checks. This worked quite well, and we found two such vulnerabilities: CPR-ID-2145 and CPR-ID-2146.

However, when reporting them to the vendor, we found out they were both already reported, by two separate groups. As these are duplicates, they should be credited to their rightful researchers, even if we submitted them only a few hours later.

We therefore decided it is more appropriate to let the other teams present their findings, and removed the details about them from our blog post.

We need a memory corruption…

At this point, we found 5 vulnerabilities that could serve as Information Disclosure exploit primitives in our attack. However, we've yet to find even a single Memory Corruption vulnerability. Looking for such vulnerabilities in FreeRDP was quite annoying, as every time we had a lead, a check blocked it. Many times, this check was the patch for a vulnerability that we've reported, so we can't really complain too much.

This post in the popular Zensploitation Twitter profile (@zensploitation) pretty much summarizes our feeling at this point in our research:

Figure 8: https://twitter.com/zensploitation/status/1244598246879547393.

At this point, we decided we've come too far to simply give up. We decided to look at the guacamole-server once more, and this time we struck gold.

CPR-ID-2144 – At last, a Memory Corruption

CVE: CVE-2020-9498
File: `protocols\rdp\plugins\guac-common-svc\guac-common-svc.c`
Function: `guac_rdp_common_svc_handle_open_event()`

The RDP protocol exposes different "devices" as separate "channels", one for each device. These include the `rdpsnd` channel for the sound, `cliprdr` for the clipboard, and so on. As an abstraction layer, the channel messages support a fragmentation that allows their messages to be up to 4GB long. To properly support the `rdpsnd` and `rdpdr` (Device Redirection) channels, the developers of guacamole-server added an additional abstraction layer, implemented in the file: `guac_common_svc.c`. Figure 9 shows the fragmentation handling, as implemented in this file:

Figure 9: Handling an incoming channel fragment.

We can see that the first fragment must contain the `CHANNEL_FLAG_FIRST` fragment, and when handled, a stream is allocated according to the overall declared length of the total message.

However, what happens if an attacker sends a fragment without this flag? It seems that it is simply appended to the previous leftover stream. At this point, this looks like a very promising Dangling-Pointer vulnerability. Now we only need to check if the developers remembered to set it to `NULL` when the previous fragmented message finished its handling.

Figure 10: Releasing the used stream **without** clearing the dangling pointer.

Figure 10 clearly shows that after a fragmented message finishes the reassembly and goes on to be parsed, it is freed. And that's it. No one sets the dangling pointer to `NULL`!

A malicious RDP server could send an out of order message fragment that uses the previously freed `wStream` object, effectively becoming a Use-After-Free vulnerability. On top of that, the `wStream` is the most powerful object that we could have hoped to get for such a vulnerability, as it can be used for an Arbitrary Write if the pointer field is set to the desired memory address. On top of all that, we have a useful Information Disclosure vulnerability in the `rdpsnd` channel, right after our corrupted `wStream` object is used. With some effort, a specially crafted `wStream` object can turn our original vulnerability into a more powerful Arbitrary Read exploit primitive.

Finally, Remote Code Execution (RCE)

As described earlier, by using vulnerabilities CVE-2020-9497 and CVE-2020-9498, we managed to implement our Arbitrary Read and Arbitrary Write exploit primitives. Using these two powerful primitives, we successfully implemented a Remote Code Execution (RCE) exploit in which a malicious corporate computer (our RDP "server") can take control of the `guacd` process when a remote user requests to connect to his (infected) computer.

Figure 11: Exploit screenshot – popping a calc from the taken-over `guacd` process.

But the journey doesn't end here. The `guacd` process only handles a single connection and runs with low privileges. Traditionally, at this point we need a Privilege-Escalation (PE) vulnerability to take over the entire gateway. And indeed, during the coordinated disclosure with Apache, one of the questions the maintainers asked was if this attack scenario was really possible. Could we somehow take over all of the connections in the gateway from only a single `guacd` process?

Let's find out.

> *"The only person in computing that is paid to actually understand the system from top to bottom is the attacker."* (Halvar Flake, offensivecon 2020)

Apache Guacamole – Deep Dive

If we dive into the network architecture of the Guacamole gateway that we saw earlier, we see the following:

Figure 12: Focused view of the Apache Guacamole architecture.

For the Privilege Escalation our focus is on these two components:

- guacamole-client – Marked as **Web Server**.
- guacamole-server – Marked as **Proxy**.

guacamole-client

The guacamole-client component is responsible for the web server that performs user authentication. This web server holds the configurations needed for each user session, storing information such as:

- Wanted protocol – Usually RDP.
- IP address of the worker's PC inside the network.
- Etc.

After a client is successfully authenticated, the guacamole-client initiates a Guacamole Protocol session with the guacamole-server to create a matching session for the client. This is done by connecting to the guacamole-server on TCP port 4822 (by default) on which the `guacd` process is listening.

After the session is created, the guacamole-client only relays information back and forth between the guacamole-server and the client's browser.

guacamole-server

According to Apache's documentation: "guacd is the heart of Guacamole." Upon startup, `guacd` listens on TCP port 4822 and waits for incoming instructions from the guacamole-client. It is important to note that the communication on this port uses no authentication or encryption (SSL could be enabled, but it isn't the default). For this reason, **we** added the two firewalls in Figure 12, which *should* be responsible for limiting the access to this TCP port, allowing only the guacamole-client to connect.

When a connection is made, `guacd` creates a new thread and invokes the function that is responsible for initiating the Guacamole Protocol. At this point, there are two user options:

- Create a new connection.
- Join an existing connection.

Side Note: We use the term **connection** instead of **session**, as this is the term used by Guacamole to refer to a connection with a given computer. Every computer has a single connection, and multiple users can share the same connection. There is no "user session" as the entire design is based on the Guacamole connection to a given computer, and users simply join connections.

The first option is by far the most widely used. In this case, a random unique id (UUID) is generated for the newly created connection, and a `fork()`ed process is spawned for it. The mapping between the UUID and the new process is stored in an in-memory dictionary called `proc-map` and the UUID is sent back to the guacamole-client. It is important to note that the spawned process immediately drops its permissions, before it initiates the connection with the computer inside the network.

The second option is quite unique, and was probably implemented so that multiple users could share a single connection and work together. In this scenario, a user requests to join an existing connection by supplying the connection's UUID. To distinguish between the users, the user who created the connection is the "owner" and other users have "owner" set to `false`. This option also includes the possibility for a read-only connection for users that are not marked as "owner."

Figure 13: Add a new user and store the process in the `proc-map` to allow others to join.

To support joining users, the spawned process for a given connection inherits a socket-pair for communicating with the parent `guacd` process. While the main thread initializes the wanted client, for example, FreeRDP for an RDP connection, another thread waits for messages from the parent process, signaling our process that a new user asked to join the connection.

The `guacd` process acts as the connection manager that spawns per-connection processes, and at the same time also implements the core logic for these spawned processes. For this reason, from now on we will refer to the parent `guacd` process as the **primary** process.

Privilege Escalation – Step-by-Step

Step #0 – Take over a single guacd process

We already have a working exploit for this part.

Step #1 – Masquerade as the guacamole-client

While the `guacd` process that we have control over is only a low-privileged process that is running inside the gateway, it still has a few useful privileges. For starters, running on the gateway enables us to connect to the primary process over TCP port 4822. As the primary process expects no authentication over this port, nothing stops us from connecting to it and controlling the process exactly like a normal guacamole-client does.

Step #2 – Harvest secrets from our memory

This is the key design choice for us to exploit. As the `guacd` executable contains the logic for both the primary process and the per-connection processes, when a new connection process is spawned, only `fork()` is used. This statement bears repeating: **Only** `fork()` is used, **without** using `execve()`!

What does this mean? A forked process contains the entire memory snapshot of its parent, and that snapshot is replaced with a new image when `execve()` is called. Without this crucial call, the child process inherits the entire memory address space of its parent. This includes:

- Full memory layout – Useful for bypassing ASLR when we want to attack the parent process.

- Full memory content – Every secret stored in the primary process is given to the child process as well.

This means that our process has the `proc-map` mapping that maps between each secret connection UUID to its respective process. We only need to find this data structure in our memory and we will have all the currently active UUIDs.

Locating proc-map itself was purely technical. In our exploit, we found it by reading our proc's memory layout file from `/proc/<pid>/maps`. The data structure is so big it was `mmap()`ed to a standalone memory allocation, and so it has its own entry in the file.

Step #3 – Join all the sessions

We already convinced the primary process that we could initiate Guacamole Protocol requests, and now we even know which requests to ask for. Our next step is to request to join each and every one of the existing connections, by supplying their now-known UUIDs.

Figure 14: Log entry marking we successfully joined an existing connection.

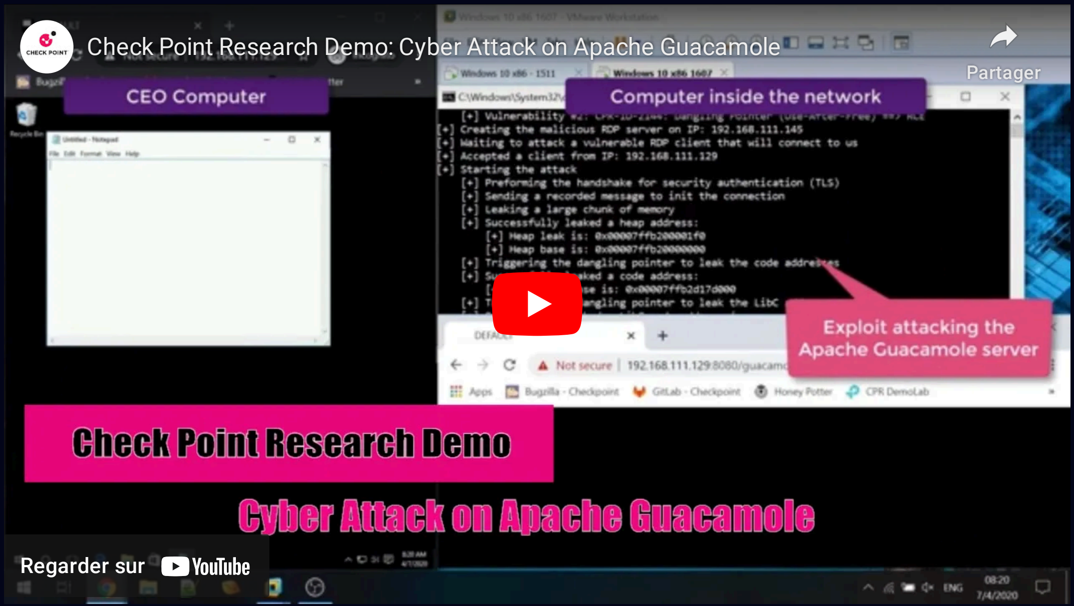
Surprisingly, the "read-only" session attribute is set by the guacamole-client. This means that although we aren't the owners of the connections, we can still turn off the "read-only" privilege bit for our joining user and get full permissions for the connection. In addition, aside from a log message in the primary process, as seen in Figure 14, there is no visible indication that another user just joined the connection.

Step #4 – Repeat

If you were paying close attention, you may have noticed a drawback in our attack plan: our `guacd` process has an outdated image of the `proc-map` mapping. Any session that begins after we were spawned would only be updated in the real `proc-map`, and therefore won't be available in our outdated memory image.

This drawback has a straightforward solution. Every chosen interval, say 5 minutes, we can send the primary process the command to start a new RDP connection and connect to our infected machine inside the network. This way, a new `guacd` is spawned, and now holds an updated version of `proc-map`. Using our original exploit, we can attack this process as well, and "refresh" our version of `proc-map`.

When chained together, here is the full exploit chain, RCE + PE, in action:



Disclosure Timeline

- 31 March 2020 – Vulnerabilities were disclosed to Apache.

- 31 March 2020 – Apache responded and requested more information.
- 31 March 2020 – Vulnerabilities were disclosed to FreeRDP.
- 31 March 2020 – FreeRDP responded and requested more information.
- 31 March 2020 – FreeRDP notified us that both CPR-ID-2145 and CPR-ID-2156 are duplicates, as they were already reported separately on 30 March 2020. (Our tough luck.)
- 08 May 2020 – Apache patched the vulnerabilities in a silent commit pushed to their GitHub.
- 10 May 2020 – We notified Apache that their patch fixes all of the reported vulnerabilities.
- 12 May 2020 – Apache issued 2 CVE-IDs to the 4 reported vulnerabilities.
- 28 June 2020 – Apache released an official patched version – 1.2.0.

Conclusions

We demonstrated a new angle to the Reverse RDP Attack scenario, an attack scenario we originally presented at the beginning of 2019. While users usually think of an RDP client… well… as a client, the Apache Guacamole scenario teaches us otherwise. In the standard case, a vulnerability in a client can be used by an attacker to take control of a single corporate computer. However, when deployed within a gateway, such vulnerabilities have a much more severe impact on the organization.

While the transition for remote work from home is a necessity in these tough times of the COVID-19 pandemic, we can't neglect the security implications of such remote connections. Using Apache Guacamole as our example target, we were able to successfully demonstrate how a compromised computer inside the organization can be used to take control of the gateway that handles all of the remote sessions into the network. Once in control of the gateway, an attacker can eavesdrop on all incoming sessions, record all the credentials used, and even start new sessions to control the rest of the computers within the organization. When most of the organization is working remotely, this foothold is equivalent to gaining **full control over the entire organizational network**.

We strongly recommend that everyone make sure that all servers are up-to-date, and that whatever technology used for working from home is fully patched to block such attack attempts. In our case, within 24 hours from finding the vulnerabilities and proving they are indeed exploitable, we implemented the security fix and became the first production environment to be secured against this security vulnerability, thus ensuring that our employees can safely connect remotely.

Recommendation for Protection

Check Point's IPS blade provides protection against this threat:
"Apache Guacamole Remote Code Execution"

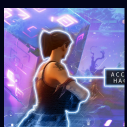
GO UP

BACK TO ALL POSTS

POPULAR POSTS

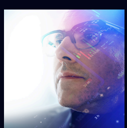
ARTIFICIAL INTELLIGENCE | CHATGPT | CHECK POINT RESEARCH PUBLICATIONS

OPWNAI : Cybercriminals Starting to Use ChatGPT



CHECK POINT RESEARCH PUBLICATIONS | THREAT RESEARCH

Hacking Fortnite Accounts



ARTIFICIAL INTELLIGENCE | CHATGPT | CHECK POINT RESEARCH PUBLICATIONS

OpwnAI: AI That Can Save the Day or HACK it Away

BLOGS AND PUBLICATIONS

CHECK POINT RESEARCH PUBLICATIONS | GLOBAL CYBER ATTACK REPORTS | THREAT RESEARCH