CVE-2020-8442: analysisd rootcheck decoder: heap overflow in DB_File. #1820

Closed cpu opened this issue on Jan 15, 2020 · 1 comment · Fixed by #1825

cpu commented on Jan 15, 2020 • edited Contributor

The `ossec-analysisd` rootcheck decoder (`src/analysisd/decoders/rootcheck.c`) allocates two fixed size heap buffers via global static vars. One, `rk_agent_ips` is an array of `*char` size `MAX_AGENTS` . The other `rk_agent_fps` is an array of `*FILE` , also size `MAX_AGENTS` . In a default build MAX_AGENTS is 2048.

ossec-hids/src/analysisd/decoders/rootcheck.c
Lines 21 to 22 in abb36d4

```
21      static char *rk_agent_ips[MAX_AGENTS];
22      static FILE *rk_agent_fps[MAX_AGENTS];
```

When processing rootcheck messages the `RK_File` function is called to find a file pointer for the given agent name.

In `RK_File` a `while` loop with a index counter `i` is used to try and find an index of `rk_agent_ips` that matches the provided `agent` name.

No check of `i` is made to ensure that it stays within the bound of `MAX_AGENT` , resulting in a straight-forward heap buffer overflow when more than `MAX_AGENT` syscheck update messages for distinct agent names are processed.

This code was introduced in the original `rootcheck` functionality with `5546ed6` on Oct 9, 2005. I believe it affects OSSEC v2.7+.

This is triggerable via an authenticated client through the `ossec-remoted` . The client needs only write MAX_AGENT rootcheck update messages with distinct message agent names.

While `ossec-remoted` always sets the agent name portion of messages passed on to `ossec-analysisd` with a prefix out of the attackers control based on the agent key and src IP (`($NAME) $SRCIP->`) the portion after this prefix is attacker controlled and thus can be mutated to make more than MAX_AGENT unique names that will be decoded by the `ossec-analysisd` .

Notably this bug has fairly high potential for exploitation. The attacker is able to overwrite a *FILE pointer with a pointer to the agent name, which is mostly attacker controlled (minus a short prefix), and can be up to 255 - strlen(prefix) bytes long. I'm definitely able to reliably segfault the `ossec-analysisd` process with this bug though I was personally unable to achieve control of execution.

Overwriting a *FILE pointer with a pointer to attacker controlled data is a common way to achieve reliable code execution. There are many pointers in the FILE struct to be abused and while libc has added some hardening it isn't applicable for versions 2.24 or lower (e.g. Ubuntu 16.04) and bypass techniques are well known.

https://outflux.net/blog/archives/2011/12/22/abusing-the-file-structure/
https://dhavalkapil.com/blogs/FILE-Structure-Exploitation/
https://www.slideshare.net/AngelBoy1/play-with-file-structure-yet-another-binary-exploit-technique

In this case there are two additional challenges to exploiting the bug that stumped me but may not stump someone who is actually good at writing exploits :-)

1. Since the protocol is all string based you can't use a null byte in payload that overwrites the *FILE contents which makes specifying valid pointers on x86_64 very challenging. (You get one terminating \0 at the end of your payload which can be used for the high order byte of a pointer, but it's still a challenge to find useful targets in high mem).
2. Getting your overwritten `rk_agent_fps` entry used with `fseek` requires being able to specify the `rk_agent_ips` name at the matching `i` value. Usually this is the first bytes of a valid `FILE` and so I *think* the attacker needs full control of the agent name to be able to specify a match (usually `\200,\255\373` or similar. The value is predictable based on rootcheck's `open` flags). If I'm correct this might mean the segfault is remotely triggerable but exploiting the FILE overwrite may not be.

To fix this the `while` conditions in `DB_File` and `DB_SetCompleted` should be rewritten to short circuit if `i >= MAX_AGENTS` :

E.g. instead of:

```
while (i < MAX_AGENTS) {
```

Use:

```
while (i < MAX_AGENTS && rk_agent_ips[i] != NULL) {
```

I think it's worth noting that this bug is nearly identical to the one reported by Paul Southerington in the syscheck decoder, patched in Feb 2012: `91aa29a`

It looks like this was fixed in Wazuh's OSSEC fork at some point, though potentially without realizing it fixed a vulnerability:
https://github.com/wazuh/wazuh/blob/413b72b17070350b62b2176c2bdc310cc66d30f6/src/analysisd/decoders/rootcheck.c#L77

This seems like a place where process improvement could help. Receiving vulnerability reports should trigger a search through the codebase for equivalent problems.

This was referenced on Jan 15, 2020

OSSEC-HIDS Security Audit Findings #1821
Closed

analysisd: fix heap overflow in rootkit decoder. #1825
Merged

ddpbsd closed this as completed in #1825 on Jan 16, 2020

cpu changed the title ~~analysisd rootcheck decoder: heap overflow in DB_File.~~ CVE-2020-8442: analysisd rootcheck decoder: heap overflow in DB_File. on Jan 30, 2020

cpu commented on Jan 30, 2020 Contributor Author

This was assigned CVE-2020-8442

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

Successfully merging a pull request may close this issue.

analysisd: fix heap overflow in rootkit decoder.
cpu/ossec-hids

1 participant