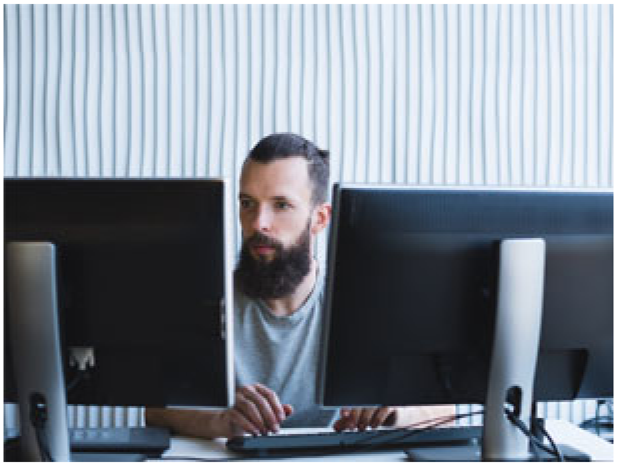
Analyzing ELF/Sshdinjector.A!tr with a Human and Artificial Analyst

fortinet.com/blog/threat-research/analyzing-elf-sshdinjector-with-a-human-and-artificial-analyst

February 4, 2025



Article Contents

By Axelle Apvrille | February 04, 2025
Affected Platform: Linux
Impacted Users: Linux-based network appliances or IoT
Impact: Data exfiltration
Severity Level: Medium

ELF/Sshdinjector.A!tr is a collection of malware that can be injected into the SSH daemon. Samples of this malware collection surfaced around mid-November 2024. While we have a good amount of threat intelligence on them (e.g., they are attributed to the **DaggerFly** espionage group), nobody seems to have looked into what they actually do. In this blog post, we will focus on the reverse engineering of the attack's binaries and how this reverse engineering was achieved.

FortiGuard Labs Outbreak Alerts

Subscribe today to have threat alerts delivered to your inbox

Reverse Engineering of ELF/Sshdinjector.A!tr

The attack uses several binaries:

- A dropper checks if the host is infected. If not, it drops all malicious binaries (see Figure 1) at the right places.

- A malicious SSH library named *libsshd.so* communicates with a remote bot master and will typically exfiltrate information.
- Several other infected binaries (*mainpasteheader*, *selfrecoverheader,…*) ensure the host remains infected (malware persistence).

Figure 1: Overview of ELF/Sshdinjector
More precisely, the dropper checks if it is being run under root privileges and, if not, exits. It then checks whether the host is infected by searching for a file named */bin/lsxxxssswwdd11vv containing* the word *WATERDROP*. If the host is not yet infected, it attempts to overwrite the legitimate binaries *ls*, *netstat,* and *crond* with infected binaries

(*/bin/lsxxxssswwdd11vv*, *selfrecoverheader*, *mainpasteheade,r*…). Finally, it searches the SSH daemon and, if necessary, infects it with *libsshd.so*.

The malicious payload is contained in *libsshd.so*. The main element is found in a function named “ *haha*.” It also creates two other threads from the functions “*heihei*” and “*xixi*”. Those three names refer to laughing in Chinese. Function “*xixi*” checks whether it has access to */root/intensify-mm-inject/xxx*, in which case it will kill and restart both the SSH and Cron daemons. Function “*heihei*” connects to the remote C2 (hard-coded IP address 45.125.64[.]200, port 33200 or 33223) and listens for incoming commands.

Command Id	**Description**
1	SERVER_REQ_BASE_INFO. Exfiltrates uname, MAC address etc to C2
2	List running services, by listing files in */etc/init.d*
3	Reads users from */etc/shadow*
4	Lists running process
5	Tests access to */var/log/dmesg*
6	Tests access to */tmp/fcontr.xml*
7	Lists a given directory

8	File transfer
9	Opens a shell terminal
10	Executes a command in the terminal
11	Unloads and exits the malicious process
12	Removes a file
13	Renames a file
1000	SERVER_RET_ONLINE_ACK
0x80000001	Client status change notification. It sends base info, service list, read */etc/shadow*.

Communication with the C2 uses its own protocol. All packets include a hard-coded UUID (*a273079c-3e0f-4847-a075-b4e1f9549e88*), an identifier (*afa8dcd81a854144*), and the response to the command.

AI-Assisted Malware Analysis

Reverse engineering was performed using Radare2, assisted by Generative AI through the Radare2 extension "r2ai."

This study shows that AI provides excellent insights into the malware, delivering high-quality source code that complements the output obtained from a standard decompiler.

For example, I used **r2ai in "auto" mode**. In this mode, the user asks the AI a question, and the AI automatically performs the necessary steps with the radare2 disassembler to answer. This is particularly helpful for users who don't know Radare2 well.

Figure 2: r2ai runs in auto mode and automatically issues r2 command "iz" to start working on the question.
In this screenshot, we see the AI automatically searches for strings in the binary, via r2's command "iz".

The overview of the dropper is excellent. **AI excels in reading large quantities of information and summarizing them.**

Figure 3: The AI summarizes quite well the behavior of the malware.
We can then ask the AI to decompile the main. **While the AI-generated source code is easy to understand, its details are not always correct**. By comparison, source code produced by decompilers is often difficult to read but is accurate. Because of this, it is important to remember that these approaches complement each other and are ideally viewed side by side.

Figure 4: This source code was generated by the AI, via r2ai. It is globally correct, readable and useful. Only comments marked “AXELLE REMARK” are my own, and highlight a few errors of the AI.
While AI performs very well, there are many cases in which it does not produce a satisfactory answer, at least not at first.

The most common issue is **hallucination**, wherein the AI *invents* something that isn’t true. Worse, it’s not always easy for a human analyst to spot hallucinations because the AI can sound very convincing. For example, in the AI-generated code below, the AI completely created an upload and a download command that is pure invention.

Figure 5: An example of AI hallucination: the botnet does not have any FILE_DOWNLOAD nor FILE_UPLOAD command. This is an invention, misunderstanding an existing “file copy” feature.
Another frequent issue is **extrapolation**, where the AI does not totally invent something but *extrapolates* it. For example, the AI says the malware “manipulates” the MAC address. This is far-fetched. While it *creates* a string containing the host's MAC address and exfiltrates it, there is no modification of the MAC address. In a related example, the AI claims the malware hides its network communications. It does not. It hides on the OS by infecting common binaries such as netstat, but it does not attempt to hide the communication itself.

Figure 6: AI extrapolation. The sentences outlined in red have been largely exaggerated by the AI.
Yet another issue is **omissions**. Omissions are the downside of AI’s power to summarize situations. Its summaries often lack the details a human would find important. For example, at some point, the malware tests access to a file named */tmp/fcontr.xml*. Despite this being absolutely clear in the assembly, **the first version of AI-generated code completely eluded this part**. The solution to this issue is to **ask again by modifying the question**/prompt for the AI. In this case, I simply added to the end of the prompt: “Please pay attention to what is around fcontr.xml” and it solved the problem. Of course, this requires knowing that something had been omitted in the first place.

In fact, **interactions with AI are seldom perfect in a single shot**. Rather, they could be compared to a discussion with a capable colleague with impressive knowledge and intelligence but less intuition and experience. For this research, I kept my disassembler open. I used it several times to check for hallucinations, assist the AI (!) when it failed to find correct addresses or cross-references, or guide it to look into interesting parts.

Conclusion

While disassemblers and decompilers have improved over the last decade, this cannot be compared to the level of innovation we are seeing with AI. This is **outstanding**!

AI is particularly good at providing overviews of samples and generating easy-to-understand source code. Fortunately—or not?—**AI cannot work alone and must be piloted and complemented by competent human analysis to spot hallucinations** (the most dangerous issues), **refine questions, identify omissions, or guide** the AI in the most interesting direction.

I haven't discussed *language models yet*. Obviously, r2ai's results depend on the language model used. Language models are configurable, and we can conveniently switch from one to another, whether a local model or a remote one, free access or paid. The results from this article were mainly obtained using Claude 3.5 Sonnet 2024-10-22.

Last but not least, this blog post was written without AI assistance ;-)

Fortinet Protections

Fortinet customers are already protected from this malware variant through our AntiVirus as follows: FortiGuard Labs detects the sample with the following AV signatures:

ELF/Sshdinjector.A !tr and Linux/Agent.ACQ!tr

The FortiGuard AntiVirus service is supported by FortiGate, FortiMail, FortiClient, and FortiEDR. Fortinet EPP customers running current AntiVirus updates are also protected.

IOCs

94e8540ea39893b6be910cfee0331766e4a199684b0360e367741facca74191f

0e2ed47c0a1ba3e1f07711fb90ac8d79cb3af43e82aa4151e5c7d210c96baebb

6d08ba82bb61b0910a06a71a61b38e720d88f556c527b8463a11c1b68287ce84