Umbreon Linux Rootkit Hits x86, ARM Systems

blog.trendmicro.com/trendlabs-security-intelligence/pokemon-themed-umbreon-linux-rootkit-hits-x86-arm-systems/

September 5, 2016

The Trend Micro Forward Looking Threat Research team recently obtained samples of a new rootkit family from one of our trusted partners. We are providing a detailed analysis of the rootkit, and also making the samples available to the industry to help others block this threat. This rootkit family called Umbreon (sharing the same name as the Pokémon) targets Linux systems, including systems running both Intel and ARM processors, expanding the scope of this threat to include embedded devices as well. (An aside: the rootkit does appear to be named after the Pokémon of the same name. This Pokémon is known for hiding in the night, which is an appropriate characteristic for a rootkit.) We detect Umbreon under the ELF_UMBREON family. The development of Umbreon began in 2011, and we've seen discussions about it in the cybercriminal underground since at least 2013. It has been claimed in underground forums and IRC channels by several underground actors that Umbreon is very hard to detect. Our research shows how this rootkit works, and how it is tries to achieve stealth within a Linux environment. Umbreon is manually installed onto an affected device or server by the attacker. This can be done either physically or remotely (if the attacker has obtained remote access to the device). Once installed, it can be used by the attacker to take control of the affected device. ***What is a ring 3 rootkit?*** Rootkits are persistent threats intended to be hard to detect/observe. Its main purpose is to keep itself (and other malware threats) stealthed and totally hidden from administrators, analysts, users, scanning, forensic, and system tools. They may also open a backdoor and/or use a C&C server and provide an attacker ways to control and spy on the affected machine. There are various execution modes where code can run, with corresponding levels of access. These are:

- User mode (ring 3)

- Kernel mode (ring 0)
- Hypervisor (ring -1)
- System Management Mode – SMM (ring -2)

Research on running rookits within certain chips on motherboards or other devices has been carried out; such a rootkit would run in ring -3. The lower the level a piece of code runs, the harder it is to detect and mitigate. However, this does not mean a ring 3 rootkit is simple or easy to remove. A ring 3 rootkit (or usermode rootkit) does not install kernel objects onto the system, but hooks functions from core libraries that are used by programs as interfaces to system calls that run important operations in a system such as reading/writing files, spawning processes, or sending packets over the network. It is perfectly possible to spy on and change the way things are done within an operating system, even from user mode. On Linux, when a program calls the *printf()* function there are other cascaded functions called by this one like *_IO_printf()* and *vprintf()* that are in the same library. All of these end up calling the *write()* system call. While a ring 0 rootkit would hook this system call in kernel mode (and this require the insertion of a kernel object/module into the system), a ring 3 rootkit would hook one of the intermediary library functions in userland, removing the need for native code in the kernel (something which would be fairly difficult to achieve). ***Cross-platform features*** We were able to successfully get Umbreon running on three different platforms: x86, x86-64 and ARM (Raspberry Pi). The rootkit is very portable because it does not rely on platform-specific code: it is written in pure C, except for some additional tools that are written in Python and Bash scripting. Our analysis indicates that this was by design: Umbreon's did this intentionally so that it could easily support the three platforms noted above. ***Backdoor authentication*** During installation, Umbreon creates a valid Linux user that the attacker can use with a backdoor into the affected system. This backdoor account can be accessed via any authentication method supported by Linux via pluggable authentication modules (PAMs), including SSH. This user has a special GID (group ID) that the rootkit checks to see if the attacker is attempting to access the system. It is not possible to see this user in files like */etc/passwd* because *libc* functions are hooked by Umbreon. The picture below shows the welcome screen shown when this backdoor account is accessed via SSH:

Figure 1. SSH login screen (Click to enlarge)

Espeon backdoor component This is a non-promiscuous *libpcap*-based backdoor written in C that spawns a shell when an authenticated user connects to it. (The attackers also named this component after a Pokémon - this time Espeon, which has pronounced ears.) It can be instructed to establish a connection to an attacker machine, functioning as a reverse shell to bypass firewalls. Espeon captures all TCP traffic that reaches the main Ethernet interface of the affected computer. Once it receives a TCP packet with some special field values, it then connects back to the source IP of this TCP packet. These are the values that Espeon watches out for:

- Sequence number (SEQ) is 0xc4
- Acknowledgement number (ACK) is 0xc500

- IP Identification (ID) is 0x0fb1

These conditions would all be set by the attacker in a packet he would send to the affected machine. If all three values match, the backdoor connects back to the sending IP address. Here is the disassembled of *got_packet()* function, where this comparison is performed:

Figure 2. Code sample (click to enlarge)

Hiding pre-loaded configuration files from system call tracing System call tracing is a technique used by a very popular Linux command line tool called *strace.* It uses the *ptrace()* syscall to inspect the syscall parameters and return values of other executable files. As Umbreon uses an */etc/ld.so.<random>* file to instruct the loader to load itself before any other library used by ELF binaries, it can disguise itself from administrators that use *strace* by hooking *vprintf()*, *__vfprintf_chk(),* and *fputs_unlocked().* These are used by different versions of *strace* to write to a given file descriptor. The following screenshot shows the code that does this for *vprintf()* in the *strace.so* component:

Figure 3. Code sample(click to enlarge)

wrapper_200da0_6b0 ends up in the *strstr()* function. Here, the pre-loaded configuration file is */etc/ld.so.NfitFd2* so if any argument passed to *vprintf()* function contains this text, it will be replaced by */etc/ld.so.preload.* An analyst may then believe that this is the file being used by the loader. The screenshot below shows the strings used by this routine:

Figure 4. Code sample

This component also unsets the *LD_PRELOAD* environment variable so analysts can't hook the malicious functions. ***Hiding packets*** Umbreon also hooks the *libpcap* functions *got_packet()* and *pcap_loop()* and prevents them from returning any information about TCP packets with ports between the lowest port number and highest port number specified in its configuration file. An analyst capturing network traffic with tools like *tcpdump* on the machine wouldn't be able to capture any backdoor traffic. ***Umbreon's implementation*** Umbreon acts as a library that imitates the *glibc* (GNU C Library). It creates a file called */etc/ld.so.<random>* that, according to the official documentation, has the following function:

> ```
>  /etc/ld.so.preload              File containing a whitespace-separated
> list of ELF shared objects to be loaded before the program.
> ```

Originally, the ELF loader will look for a */etc/ld.so.preload* file as the documentation clearly states. However, we found that Umbreon also patches the loader library (*/lib/x86_64-linux-gnu/ld-2.19.so* as an example) to use */etc/ld.so.<random>* instead, where *<random>* is a 7-character-string, matching the length of "preload". Every library path in this file will be loaded

before any other ELF program is launched. Inside this file, Umbreon puts the path for its own main library, which contains lots of functions matching the names
of *glibc* functions. The location of this main library is:

/usr/share/libc.so.<random>.${PLATFORM}.ld-2.22.so

${PLATFORM} is replaced by the loader with one of the following highlighted values, depending on the target architecture:

- /usr/share/libc.so.<random>.**v6l.ld**-2.22.so (for ARM)
- /usr/share/libc.so.<random>.**x86_64**.ld-2.22.so (for x86-64)
- /usr/share/libc.so.<random>.**i686**.ld-2.22.so (for x86)

However, because Umbreon is manually installed onto a compromised machine, this default path may vary. The functions hooked and implemented by the main Umbreon library are:

- __fxstat
- __fxstat64
- __lxstat
- __lxstat64
- __syslog_chk
- __xstat
- __xstat64
- access
- audit_log_acct_message
- audit_log_user_message
- audit_send
- chdir
- check_and_fix_ldso
- checkpw
- chmod
- chown
- cleanup
- dlinfo
- dlsym
- esh
- execve
- execvp
- fake_preload_fail
- fchmod
- fchown
- fchownat
- fdopendir
- filesize
- find_dlsym
- find_sym

- fopen
- fopen64
- fstat
- fstat64
- get_hideports
- get_my_procname
- get_procname
- getpath
- getpgid
- getpriority
- getpwnam
- getpwnam_r
- getpwuid
- getsid
- getspnam
- is_dynamic
- is_hideport
- is_ldso32
- is_ldso64
- is_really
- kill
- lchown
- link
- login
- lstat
- lstat64
- netstat
- open
- open64
- openat
- opendir
- pam_acct_mgmt
- pam_authenticate
- pam_open_session
- pam_prompt
- pam_vprompt
- pcap_loop
- procstatus
- procstatus_o
- pututline
- pututxline
- rclocal

- read
- readdir
- readdir64
- readlink
- recover_dirname
- recover_filename
- reinstall_self
- rename
- rmdir
- sched_getaffinity
- sched_getparam
- sched_getscheduler
- sched_rr_get_interval
- setgid
- setregid
- setresgid
- socket
- spoof_maps
- spoof_smaps
- stat
- stat64
- sxor
- symlink
- sysinfo
- syslog
- unfuck_linkmap
- unlink
- unlinkat
- write

Many of these function names match existing *libc* function names. As Umbreon's library is loaded before any other user library when an executable in launched, the loader will resolve these functions' addresses instead of the ones in *libc*. This way ,an executable will call the malicious functions invisibly. These malicious functions then inspect the arguments they receive before calling the real ones. Similarly, the output of every command may have been modified before the user sees it. It effectively functions as an in-the-middle attack, modifying both the input and output of system functions. Users cannot trust the outputs of system commands like *ps, ls, top,* and *pstree* (among others). Because they all use these *libc* functions, they will all produce modified outputs. ***How to detect Umbreon*** Most of the tools you will find in Linux are written in C. Even programs written in Perl, Python, Ruby, PHP and other scripting languages end up calling GNU C Library wrappers as their interpreters are also written in C. Because Umbreon library hooks *glibc* functions, creating a reliable tool to detect Umbreon would require a tool that doesn't use *glibc*. One way is to develop a small tool to list the contents of the default Umbreon rootkit folder using Linux kernel syscalls directly. This bypasses any malicious C library installed by Umbreon. If the output contains one or more files with names starting with *libc.so* followed by a random integer, this is the red flag that suggests Umbreon is installed in the machine. We have also created YARA rules that detect Umbreon, which can be downloaded here. ***Removal Instructions*** Umbreon is a ring 3 (user level) rootkit, so it is possible to remove it. However, it may be tricky and inexperienced users may break the system and put it into an unrecoverable state. If you are brave enough to proceed, the easiest way is to boot the affected machine with Linux LiveCD and follow the steps:

1. Mount the partition where the */usr* directory is located; write privileges are required.
2. Backup all the files before making any changes.
3. Remove the file */etc/ld.so.<random>*.
4. Remove the directory */usr/lib/libc.so.<random>*.
5. Restore the attributes of the files */usr/share/libc.so.<random>.<arch>.*.so* and remove them as well.
6. Patch the loader library to use */etc/ld.so.preload* again.
7. Umount the partition and reboot the system normally.

Here is a real-life example (please notice file names *will vary* as they are randomly chosen by the malware). In the following case, */dev/sda1* is the partition containing the */usr* directory.

```
# mount /dev/sda1 /mnt # rm -f /mnt/etc/ld.so.khVrkEQ # rm -rf
/mnt/usr/lib/libc.so.41762810374176281037/ # chattr -ai
/mnt/usr/share/libc.so.4176281037.* # rm -f
/mnt/usr/share/libc.so.4176281037.* # sed -i
's:/etc/ld\.so\.khVrkEQ:/etc/ld.so.preload:' /lib/x86_64-linux-gnu/ld-
2.19.so # umount /mnt # reboot
```

In this case, the *chattr* command is necessary because Umbreon libraries have a (append-only) and i (immutable) attributes set. ***Indicators of Compromise*** The following file samples are tied to this threat:

- b5e68f8e23115bdbe868d19d09c90eb535184acd — /.bashrc
- 73ddcd21bf05a9edc7c85d1efd5304eea039d3cb — /bin/pkg
- 48a6e43af0cb40d4f92b38062012117081b6774e — /bin/espeon-shell (detected as BKDR_UMREON.A)
- 88aea4bb5e68c1afe1fb11d55a190dddb8b1586f —/bin/unhide-self
- 73ddcd21bf05a9edc7c85d1efd5304eea039d3cb — /bin/zypper and ./bin/emerge
- 42802085c28c0712ac0679c100886be3bcf07341 — /bin/umbreon.py
- 66d246e02492821f7e5bbaeb8156ece44c101bbc — /bin/espeon (detected as ELF_UMREON.A)
- 73ddcd21bf05a9edc7c85d1efd5304eea039d3cb —/bin/yum
- 4f6c6d42bdf93f4ccf68d888ce7f98bcd929fc72 — /bin/spytty
- 73ddcd21bf05a9edc7c85d1efd5304eea039d3cb — /bin/apt-get
- 1f1ab0a8e9ec43d154cd7ab39bfaaa1eada4ad5e — /bin/.x

- 81ad3260c0fc38a3b0f65687f7c606cb66c525a8 — /.init-append
- 7b10bf8187100cdc2e1d59536c19454b0c0da46f — /.umbreon-ascii
- 96d5e513b6900e23b18149a516fb7e1425334a44 — /.profile
- 851b7f07736be6789cbcc617efd6dcb682e0ce54 — /usr/share/libc.so.2284441204.i686.ld-2.22.so (detected as ELF_UMREON.A)
- e2bc8945f0d7ca8986b4223ed9ba13686a798446 — /usr/share/libc.so.2284441204.x86_64.ld-2.22.so (detected as ELF_UMREON.A)
- 17b42374795295f776536b86aa571a721b041c38 — /.ldso/strace.so (detected as ELF_UMREON.A)
- 394fae7d40b0c54c16d7ff3c3ff0d247409bd28f —/promptlog
- 738ac5f6a443f925b3198143488365c5edf73679 —/hideports
- 022be09c68a410f6bed15c98b63e15bb57e920a9 — espeon (ARM version, detected as ELF_UMREON.B)
- 3762c537801c21f68f9eac858ecc8d436927c77a — pkg (ARM version, detected as ELF_UMREON.B)
- 2cd24c5701a7af76ab6673502c80109b6ce650c6 — strace.so (ARM version, detected as ELF_UMREON.B)
- 358afd4bd02de3ce1db43970de5e4cb0c38c2848 — umbreon.so (ARM version, detected as ELF_UMREON.B)

Update as of September 15, 2016, 8:00 PM PDT The developer of Umbreon has been in touch with us since the publication of this post. He told us that he started work on Umbreon in 2011, basing it off three existing rootkits: Jynx, Jynx2, and Azazel. All three are publicly known Linux rootkits. He has expressed his sadness and displeasure at how his code has since been abused by various malicious threat actors.