



From The Tunnels Below Gotham

Anti-Forensics The Rootkit Connection

Black Hat USA 2009
Las Vegas, Nevada

Bill Blunden

Principal Investigator
Below Gotham Labs

$$-k_B \sum_i P_i \log_e(P_i)$$



$$-k_B \sum_i P_i \log_e(P_i)$$

Introduction

The Quandary of Live Response
Another Option: Post-Mortem Analysis
Anti-Forensic Strategies

Tactics & Countermeasures

Forensic Duplication
Recovering Files
Recovering Deleted Files
Capturing a Metadata Snapshot
Identifying Known Files
File Signature Analysis
Static Analysis of an .EXE
Runtime Analysis of an .EXE

Data Source Elimination

Memory-Resident Rootkits
Firmware-Based Rootkits

Operational Issues

Footprint and Fault-Tolerance
Launching a Rootkit
Conclusions



$$-k_B \sum_i P_i \log_e(P_i)$$

Fundamental Issue → A rootkit can interfere with runtime data collection

The Athens Affair

Rootkit monitored digitized voice traffic on Ericsson AXE switches
Patched the commands that listed active code blocks
Integrity checking code was subverted (patch suspected)

<http://www.spectrum.ieee.org/telecom/security/the-athens-affair>

The DDefy Rootkit

Vendors downplay the threat to live disk imaging as unlikely
DDefy Injects a filter driver to feed bad data to forensic tools

http://www.ruxcon.org.au/files/2006/anti_forensic_rootkits.ppt

Defeating Hardware-Based RAM Capture on AMD64

Vendors attempt to sidestep OS entirely to avoid interference
Rutkowska defeated this by manipulating Northbridge map table

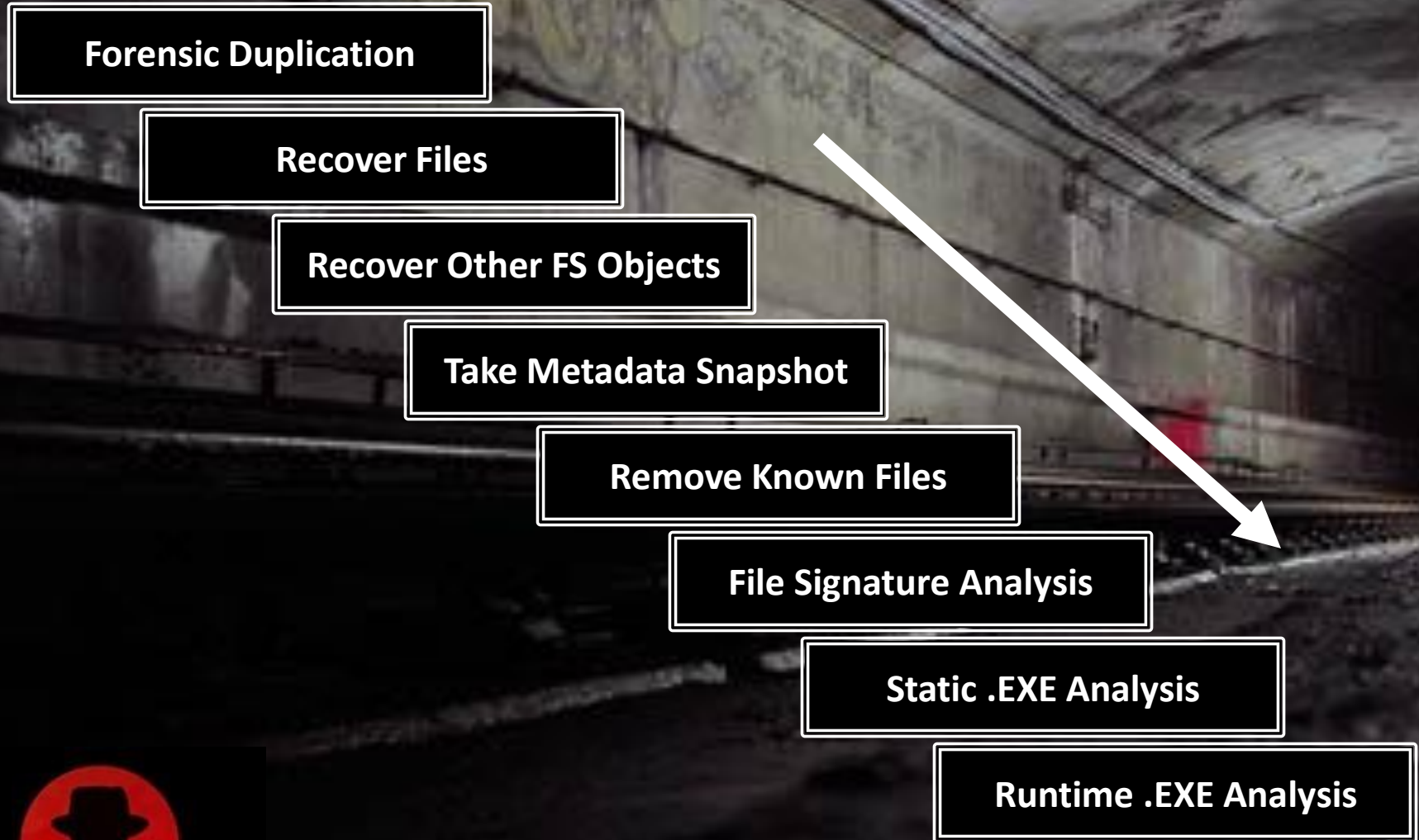
<http://invisiblethings.org/papers/cheating-hardware-memory-acquisition-updated.ppt>



Another Option: Post-Mortem Analysis

$$-k_B \sum_i P_i \log_e(P_i)$$

**Below Gotham
Laboratories**



$$-k_B \sum_i P_i \log_e(P_i)$$

Assumption

For the sake of keeping things interesting:

Let's assume we're facing off against a highly skilled, well-armed, adversary

The "they're all idiots" mentality is dangerous (don't underestimate your opponent!)

Richard Bejtlich

Director of Incident Response, GE

Former MI officer (AFCERT, AFIWC, AIA)

<http://taosecurity.blogspot.com/>



In High-Security Environments

- Compromise may be assumed a priori
- Security professionals may employ forensic analysis *preemptively*

$$-k_B \sum_i P_i \log_e(P_i)$$

Primary Goal: Outlast the investigator (exhaust their budget, e.g. *THX 1138*)

Strategy	Tactical Implementations
Data Source Elimination	Memory-Resident Code, Autonomy
Data Destruction	Data and Metadata Shredding, Encryption
Data Concealment	In-Band, Out-of-Band, & Application Level
Data Transformation	Encryption, Compression, Obfuscation
Data Fabrication	Leave False Audit Trails, Introduce Known Files

Institute Defense in Depth

Implement strategies concurrently to augment their effectiveness

Use Custom Implementations

Want to frustrate attempts to rely on automation to save time



$$-k_B \sum_i P_i \log_e(P_i)$$

Introduction

The Quandary of Live Response
Another Option: Post-Mortem Analysis
Anti-Forensic Strategies

Tactics & Countermeasures

Forensic Duplication
Recovering Files
Recovering Deleted Files
Capturing a Metadata Snapshot
Identifying Known Files
File Signature Analysis
Static Analysis of an .EXE
Runtime Analysis of an .EXE

Data Source Elimination

Memory-Resident Rootkits
Firmware-Based Rootkits

Operational Issues

Footprint and Fault-Tolerance
Launching a Rootkit
Conclusions



$$-k_B \sum_i P_i \log_e(P_i)$$

Reserved Disk Regions

One way to undermine forensic duplication is to avoid being captured on the image
Reserved regions like the HPA and DCOs were tenable hideouts (at one point in time)

Bad News

HPA/DCO-sensitive tools are now commonplace



Example: FastBloc 3 Field Edition

Write blocker that can detect and access HPAs and DCOs

<http://forensics.marshall.edu/MISDE/Pubs-Hard/FastblocFE.pdf>

$$-k_B \sum_i P_i \log_e(P_i)$$

Tactics that Hamper File Recovery

Encrypted Volumes

Nothing to carve, looks like random bytes
Plausible Deniability → Nested encrypted volumes
Conspicuous, use as part of an exit strategy

File System Attacks

Won't necessarily obstruct file carvers
Can lead to erratic behavior (do NOT want this)
Conspicuous, use as part of an exit strategy

Concealment

Definitely has potential (at least in the short-term)

In-Band Concealment
Out-of-Band Concealment
Application Layer Concealment



$$-k_B \sum_i P_i \log_e(P_i)$$

In-Band

Use regions described by the FS specification

Examples

Reserved space in file system metadata structures
Alternate Data Streams
Clusters allocated to \$BadClus

Implementations

Data Mule FS

Developed by the grugq, targets the ext2fs file system
Stores data in inode reserved space

<http://www.blackhat.com/presentations/bh-europe-04/bh-eu-04-grugq.pdf>

Issues

Surviving file system integrity checks
Allocating a sufficient amount of storage (managing many small chunks)



$$-k_B \sum_i P_i \log_e(P_i)$$

The NTFS Master File Table (MFT)

Central repository for all NTFS file system meta-data
Is a relational database consisting of a series of records
Each file/directory corresponds to one or more 1 KB records in the MFT

Hiding Data in The MFT: FragFS

Rootkit presented at Black Hat Federal 2006 by Thompson and Monroe
Identified available reserved space and slack space in MFT records

NTFS is a Licensed Specification

Microsoft provides an incomplete Technical Reference

<http://technet.microsoft.com/en-us/library/cc758691.aspx>

For (free) low-level details, we must rely on the Linux-NTFS project

<http://sourceforge.net/projects/linux-ntfs/>

Brian Carrier also wrote a book that relates many details

<http://www.digital-evidence.org/fsfa/index.html>



$$-k_B \sum_i P_i \log_e(P_i)$$

Out-of-Band

Use regions NOT described by the FS specification

Examples

The HPA, DCOs

Slack Space (file-based, partition-based, etc.)

Implementation

Metasploit's `slacker.exe`

<http://www.metasploit.com/research/projects/antiforensics/>

Issues

Finding storage space that's unlikely to be overwritten or re-allocated

Beware of slack-space wiping tools (PGP Desktop Professional 9.0.4+)

http://www.metasploit.com/research/vulnerabilities/pgp_slackspace/

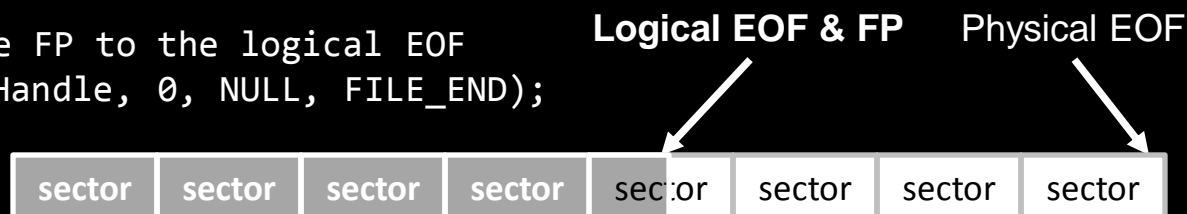


An Aside: Slacker.exe

$$-k_B \sum_i P_i \log_e(P_i)$$

Below Gotham
Laboratories

```
//Step [1] - set the FP to the logical EOF  
SetFilePointer(fileHandle, 0, NULL, FILE_END);
```



```
//Step [2] - write data between the logical EOF and physical EOF  
WriteFile(fileHandle, buffer, SZ_BUFFER, &nBytesWritten, NULL);  
FlushFileBuffers(fileHandle);
```



```
//Step [3] - move FP back to the old logical EOF  
SetFilePointer(fileHandle, -SZ_BUFFER, NULL, FILE_CURRENT);
```



```
//Step [4] - truncate the file nondestructively (on XP)  
SetEndOfFile(fileHandle);
```



$$-k_B \sum_i P_i \log_e(P_i)$$

Microsoft Addresses this Issue in Vista

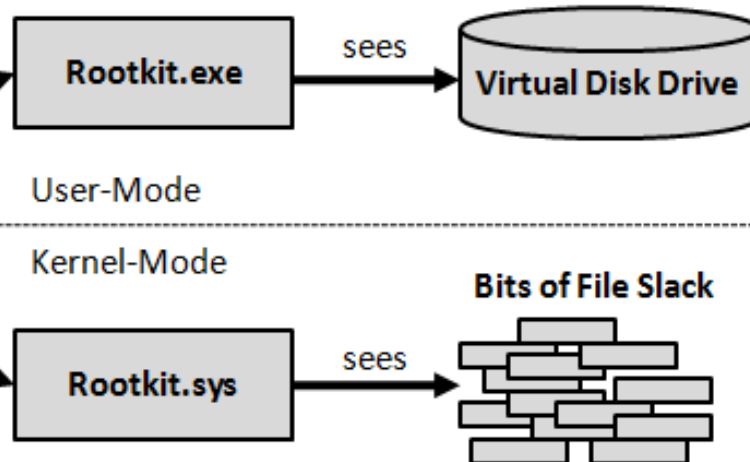
Calls to SetEndOfFile() zero out file slack space before returning

One Solution

Design a rootkit that manages file slack space from Kernel-Space
Place metadata in a known location to avoid using an external tracking file
Be Warned: don't leave this metadata in plaintext format!

KMD manages file slack space
User-Mode code sees a virtual block device

DeviceIoControl()



$$-k_B \sum_i P_i \log_e(P_i)$$

Application Layer

Use regions defined by a particular file format

Examples

Steganography

<http://www.blackhat.com/presentations/bh-usa-04/bh-us-04-raggio/bh-us-04-raggio-up.pdf>

Rogue Database Entries

Injecting code to create a Trojan Executable

Dawid Golunski, "Rogue Binaries - How to Own the Software," *hakin9*, 1/2008

Issues

Not very effective with static files, a binary diff will expose alteration

Must identify files that are normally subject to constant updates

Modifying database files through official channels leaves an audit trail

If possible, see if you can navigate the database file manually

<http://helios.miel-labs.com/downloads/registry.pdf>



$$-k_B \sum_i P_i \log_e(P_i)$$

Tactics that Impede Recovery of Deleted Data

File Wiping

Software-based wiping tools often rely on overwriting data in place
Not always effective on journaling and RAID-based file systems

Metadata Shredding

Deleting data isn't enough, must also clean up the file system
Example: The Defiler's Toolkit (TDT) built by the grugq

<http://www.phrack.org/issues.html?issue=59&id=6>

Encryption

Encrypt data before it's persisted to disk storage
Destroy the key and the data becomes random junk



How do we protect the key
From being intercepted?

$$-k_B \sum_i P_i \log_e(P_i)$$



Hints on Protecting Encryption Keys

Don't Store Keys on Disk

If you do, encrypt it with another encryption key

Minimize Runtime Key Exposure

You should assume that debuggers will be brought into play

Lock the Memory Containing the Key

Need to prevent recovery of the key from the page file/partition

On Unix: `mlock()` (see `sys/mman.h`)

On Windows: `VirtualLock()` (see `Winbase.h`)

Note: you'll need to obfuscate these calls because they're beacons

$$-k_B \sum_i P_i \log_e(P_i)$$

Tactics that Undermine the Integrity of Metadata

Binary Modification

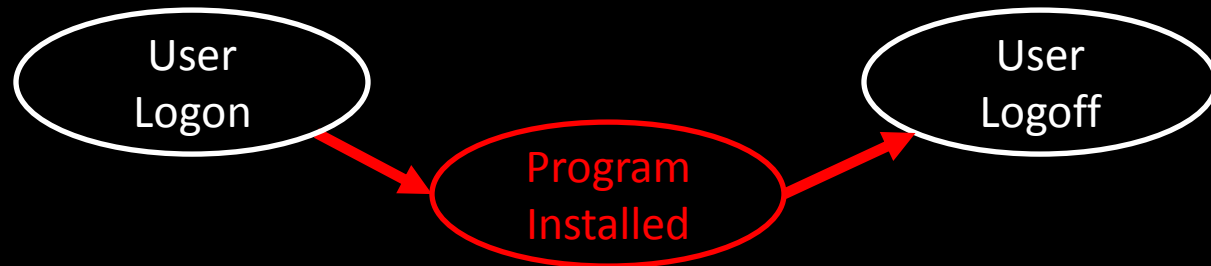
This will place a known good file into the “unknown” category
Too conspicuous for the scenario of preemptive forensics
As part of an exit strategy, serves as a diversionary measure

Timestamp Modification

Can be applied to non-system files to fabricate a false trail

Note: On NTFS, more than one attribute has timestamp data!

\$STANDARD_INFORMATION and \$FILE_NAME



$$-k_B \sum_i P_i \log_e(P_i)$$

The FILE_BASIC_INFORMATION argument stores four LARGE_INTEGER values

These values represent the number of 100-nanosecond intervals since 1601

When these values are small, the Windows API doesn't translate them correctly

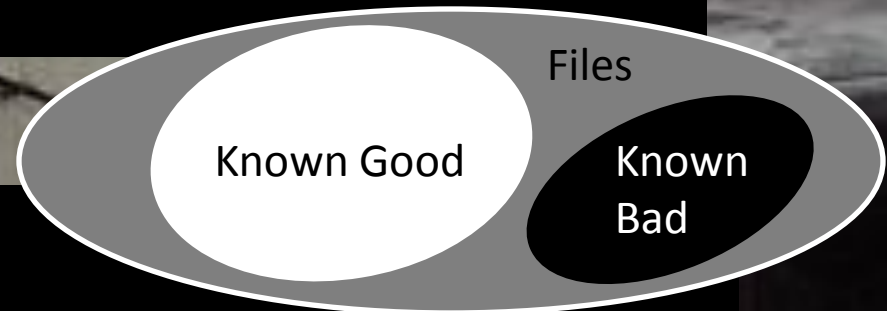
```
if(modTimeStamp)
{
    fileBasicInfo.CreationTime.LowPart = 1;
    fileBasicInfo.CreationTime.HighPart = 0;
    ...
}
ntstatus = ZwSetInformationFile
(
    handle, //IN HANDLE FileHandle
    &ioStatusBlock, //OUT PIO_STATUS_BLOCK IoStatusBlock
    &fileBasicInfo, //IN PVOID FileInformation
    sizeof(fileBasicInfo), //IN ULONG Length
    FileBasicInformation //IN FILE_INFORMATION_CLASS
);
```



$$-k_B \sum_i P_i \log_e(P_i)$$

Investigator Performs a *Cross-Time Diff*

Eliminate known good and known bad files, identify unknown files



How Can We Sabotage this Stage?

Preimage Attack

Files altered in this manner can be discovered via a binary diff

Inject Known Good and Known Bad Files

Consumes bandwidth, but is definitely conspicuous

(e.g. time needed to get reference check sums)

Has potential as part of an exit strategy

(e.g. Decrypt a known bad file, let it act as a decoy)

$$-k_B \sum_i P_i \log_e(P_i)$$

Tactics that Subvert File Signature Analysis

Transmogrification

Alter the file header so that it doesn't match the predefined signature
Keep in mind that an investigator can always crank up a hex editor

http://www.metasploit.com/data/antiforensics/BlueHat-Metasploit_AntiForensics.ppt

Steganography and Encryption

Can encrypt an executable → no signature whatsoever
Encode a configuration text file and wrap it in an executable

```
[Hidden Processes]
hxdef*
mstftp.exe
keylogger.exe
```



```
<StringData><HP>
aHhkZWYqDQptc3RmdHAuZX
h1DQprZX1sb2dnZXIuZXhl
</HP></StringData>
```



Static Analysis of an .EXE

$$-k_B \sum_i P_i \log_e(P_i)$$

**Below Gotham
Laboratories**

Static Analysis Tools	Example
File Header Readers	dumpbin.exe
Disassemblers	IDA Pro
Hex Editors	HxD

Countermeasures

Store the .EXE in a format that cannot be readily analyzed

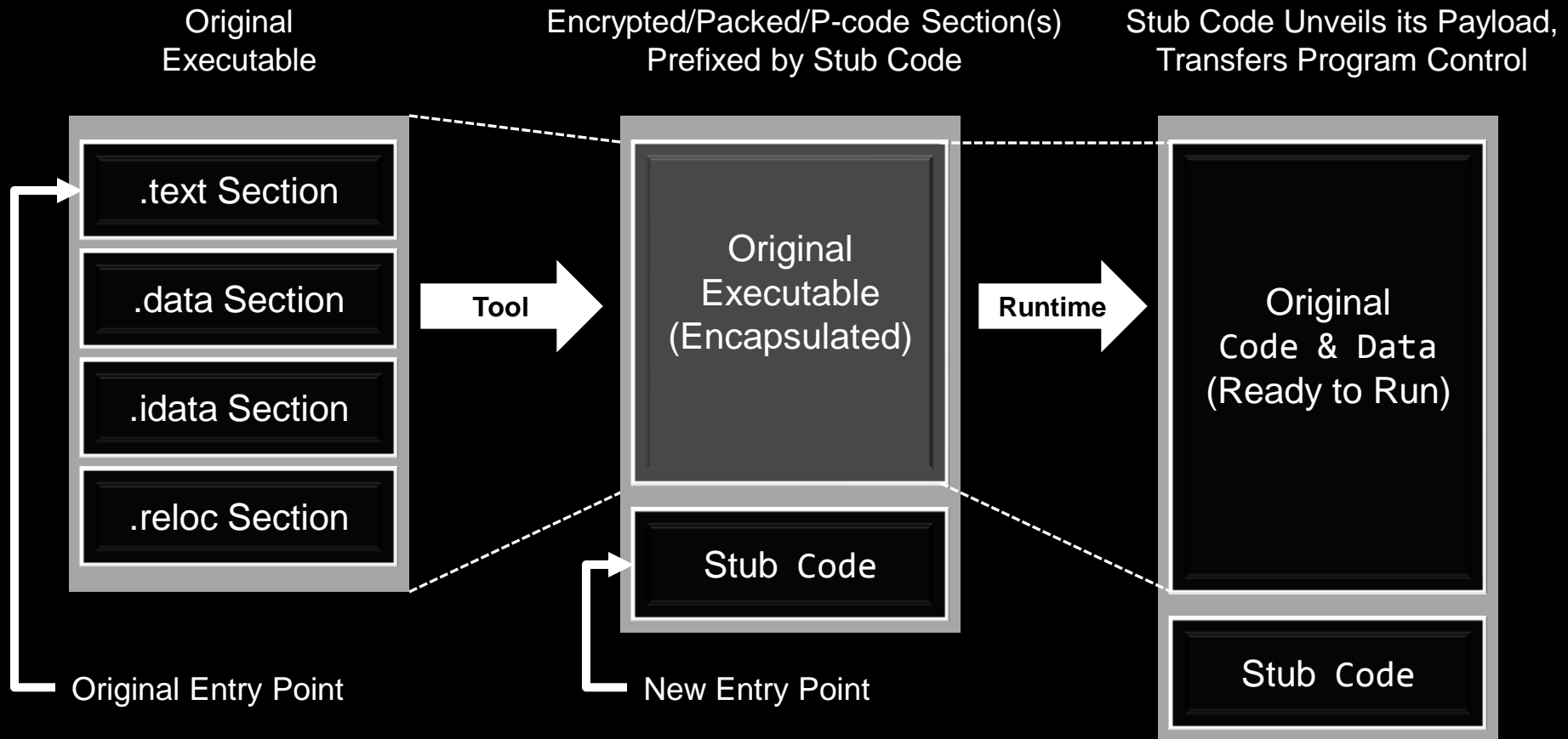
Countermeasure Tools	Description
Cryptor (e.g. EXECryptor)	Encrypts the original application
Packer (e.g. UPX)	Compresses the original application
Bytecode Compiler	Recasts the machine code as p-code



Basic Operation

$$-k_B \sum_i P_i \log_e(P_i)$$

**Below Gotham
Laboratories**



$$-k_B \sum_i P_i \log_e(P_i)$$

Origins

Standard family of exec functions on Unix systems

```
int execl(const char *path, char *const argv[]);  
int execve(const char *path, char *const argv[], char *const envp[]);  
int execlp(const char *file, char *const argv[]);
```

Replace the current process image with a new process image
Core functionality is provided by facilities in the kernel

Stub Code \approx Userland Exec



Loads an arbitrary byte stream (from disk)
Makes adjustments so that the byte stream can execute
Doesn't use the native OS loader (e.g. it's a User-Mode loader)
This sort of functionality will prove useful later on...

$$-k_B \sum_i P_i \log_e(P_i)$$

If Key Material is Stored in the Stub



Break the payload into segments, use a different key for each one
Use multiple keys that are generated at runtime from a seed value

Storing Key Material Outside of the Stub

Hide key material in a reserved region (MFT, HPA, BIOS, etc.)
Use an *environmental key*, that's specific to the target machine

<http://papers.weburb.dk/archive/00000136/01/eicar05final.pdf>

Use Custom Tools



Public tools leave a signature (<http://www.peid.info/>)
This enables automated tools that unpack/decrypt the payload
Implement a combination of packing, encrypting, and bytecode
For example: bytecode is encrypted and then compressed
Use multiple packing/encrypting algorithms to buy time
But, be aware of the size penalty you will pay

$$-k_B \sum_i P_i \log_e(P_i)$$

Camouflage

Stub code has only a few sections, not many imports, and very little string data

Section Table

01 UPX0	VirtSize: 00052000	VirtAddr: 00001000	} ← Look familiar?
02 UPX1	VirtSize: 0001B000	VirtAddr: 00053000	
03 .rsrc	VirtSize: 00008000	VirtAddr: 0006E000	

This scarcity of data is a dead giveaway to the investigator

Can Fabricate extra code and data to make the stub appear legitimate

Example: VERSIONINFO resource-definition statement

<http://msdn.microsoft.com/en-us/library/aa381058.aspx>

Runtime Exposure

Foiling static analysis is a temporary countermeasure at best

It should be used as part of a defense in depth approach

Ultimately, the stub will unveil its payload at runtime

This leads us to the next topic...



Runtime Analysis of an .EXE

$$-k_B \sum_i P_i \log_e(P_i)$$

**Below Gotham
Laboratories**

Runtime Analysis Tools	Example
Debuggers (User & Kernel-Mode)	OllyDbg, WinDBG, KD
Resource Monitors	SysInternals Suite
API Tracers	Windows Logger .exe
Network Packet Analyzers	Wireshark
System Logs	Windows Event Logs



Countermeasures

The very same tools that vendors used to defend against crackers (role reversal!)



Countermeasure	Description
Tamperproofing	Detect and respond to patching (e.g. a debugger)
Obfuscation	Make code/data difficult to interpret and reverse
Autonomy	Rely as little as possible on the official channels



$$-k_B \sum_i P_i \log_e(P_i)$$

Step 1 – Detecting Modifications

Want to know when a debugger has set a breakpoint or disabled a routine with NOPs

Official API Calls (are relatively easy to subvert)

```
BOOL WINAPI IsDebuggerPresent(); //user-mode  
BOOLEAN KdRefreshDebuggerNotPresent(); //kernel-mode
```

Checksums are a more robust approach

Avoid a centralized checksum API, implement redundant integrity checks

Create integrity checking routines to monitor your integrity checks

Plant decoy integrity checks to mislead the investigator

Periodically reinstate code to prevent it from being overwritten with NOPs

Step 2 – Responding to Modifications

Disassociate integrity checks from response (delayed trigger)

Embed subtle bugs and have the integrity checks correct them

Do NOT crash and burn, send them on a goose chase (buy time)



$$-k_B \sum_i P_i \log_e(P_i)$$

Code Morphing is preferable, has less impact on the source base
http://www.strongbit.com/execryptor_inside.asp

Obfuscation Strategy	Tactics
Reduce Code Abstraction	In-line expansion, central routine dispatching
Rearrange Code	Code interleaving
Break Conventions	Using exceptions to transfer program control
Encrypt Code	Use code checksums as a decryption key

Note - encrypted routines are inherently not thread-safe

Microsoft uses obfuscation to implement Kernel Patch Protection
<http://uninformed.org/index.cgi?v=3&a=3&p=4>

Skype also relies heavily on obfuscation to hamper reversing
<http://www.blackhat.com/presentations/bh-europe-06/bh-eu-06-biondi/bh-eu-06-biondi-up.pdf>



$$-k_B \sum_i P_i \log_e(P_i)$$

Official Channels → Windows API → Audit Trail (Event Logs)

Countermeasure

Minimize the interface between rootkit and OS
Less dependence means more stealth

User-Mode

Kernel-Mode

Rootkit

Rootkit	Implementation Details
Athens Affair	Maintained its own database instance
Deepdoor	Modified a couple of DWORDS in the NDIS data section
Deeper Door	Established a direct channel to local NIC hardware
Blue Pill	Hypervisor-based, lies outside of child partition



$$-k_B \sum_i P_i \log_e(P_i)$$

Introduction

The Quandary of Live Response
Another Option: Post-Mortem Analysis
Anti-Forensic Strategies

Tactics & Countermeasures

Forensic Duplication
Recovering Files
Recovering Deleted Files
Capturing a Metadata Snapshot
Identifying Known Files
File Signature Analysis
Static Analysis of an .EXE
Runtime Analysis of an .EXE

Data Source Elimination

Memory-Resident Rootkits
Firmware-Based Rootkits

Operational Issues

Footprint and Fault-Tolerance
Launching a Rootkit
Conclusions



$$-k_B \sum_i P_i \log_e(P_i)$$

The best way to defeat disk analysis → Never write to the disk to begin with

This strategy has so much potential that it deserves special attention

Several ways to implement



Memory-Resident Variant

Syscall Proxying

Memory-Resident Development Tools

Data Contraception

In-Memory Library Injection

Persistence by Re-Infection

$$-k_B \sum_i P_i \log_e(P_i)$$

From Earlier: Cryptors and Packers

The stub loaded a byte stream that originally resided on disk

A Full-Blown Userland Exec

Is essentially a stub that can load code from a memory buffer

The buffer usually receives its byte stream from a network connection

Sidesteps restrictions imposed by the native OS loader (e.g. disk residence)

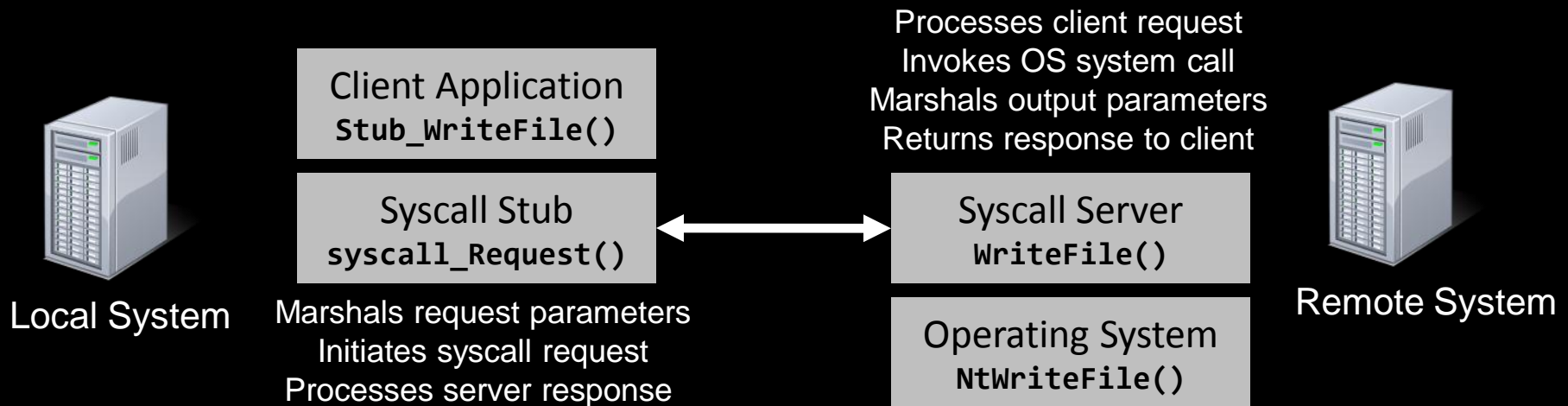
Implementations	Description
Nebbett's Shuttle	Uses Win32 API to overwrite a suspended process
ul_exec	Library that loads ELF binaries into an address space
SHELF	Revised version of ul_exec for use in exploits



Syscall Proxying

$$-k_B \sum_i P_i \log_e(P_i)$$

Below Gotham
Laboratories



Example

Core Security Technologies, CORE IMPACT Pen Testing Tool

<http://www.coresecurity.com/content/syscall-proxying-simulating-remote-execution>

Issues

Network Chatter

The average application makes lots, and lots, of system calls

Low-Level Nature of the Technique

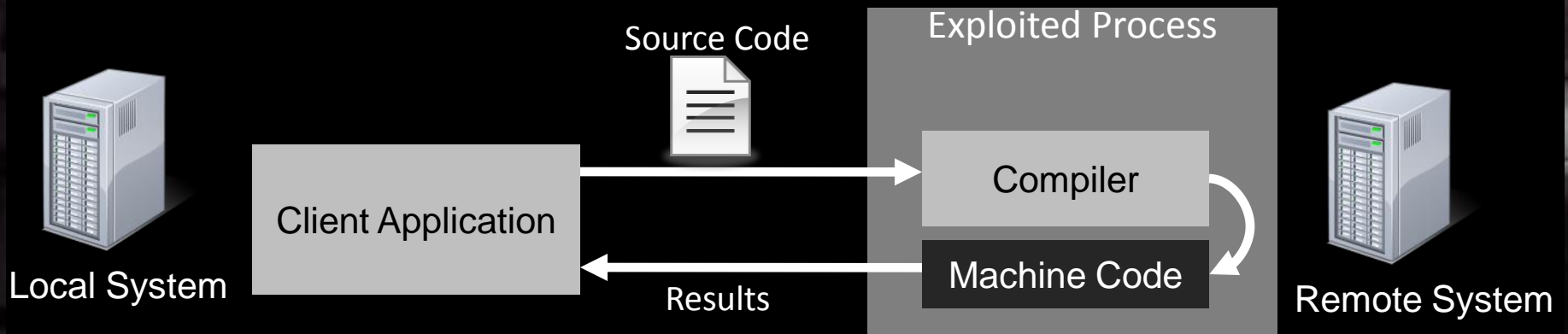
Portability becomes an issue, marshalling is a royal pain



Memory-Resident Tools

$$-k_B \sum_i P_i \log_e(P_i)$$

Below Gotham
Laboratories



Example

Immunity, Inc., CANVAS Penetration Testing Tool

Uses MOSDEF, a memory-resident C compiler that generates position independent code

<http://www.immunitysec.com/products-canvas.shtml>

<http://www.immunitysec.com/downloads/MOSDEF2dot0.tar.gz>

Variations

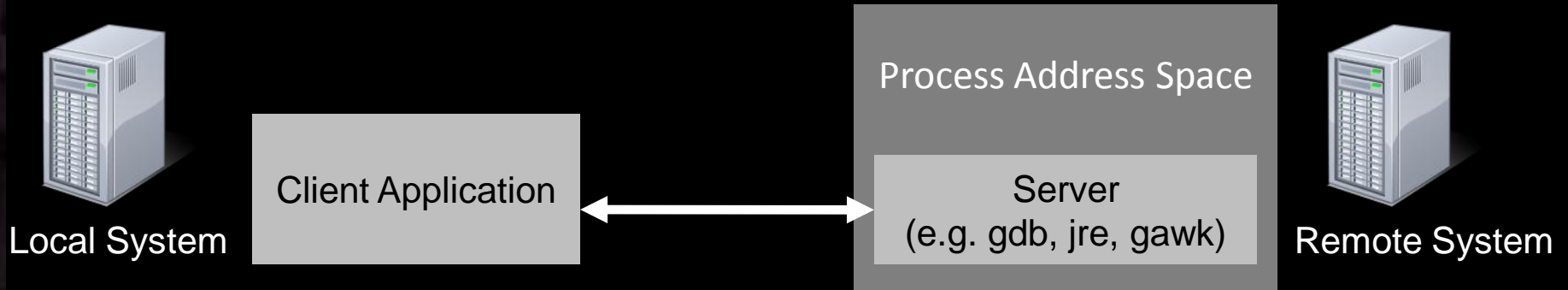
Compiler could output bytecode to be run by an *injected* virtual machine
Replace compiler by an interpreter, send it bytecode from client side



Data Contraception

$$-k_B \sum_i P_i \log_e(P_i)$$

**Below Gotham
Laboratories**



Example

Remote Exec: Built by the grugg, uses the GNU debugger and his u1_exec library

<http://www.phrack.org/issues.html?issue=62&id=8#article>

<http://archive.cert.uni-stuttgart.de/bugtraq/2004/01/msg00002.html>

Requirements

Use a Common Utility for the Server

Minimizes the amount of forensic evidence

Necessitates a "Smart" Client

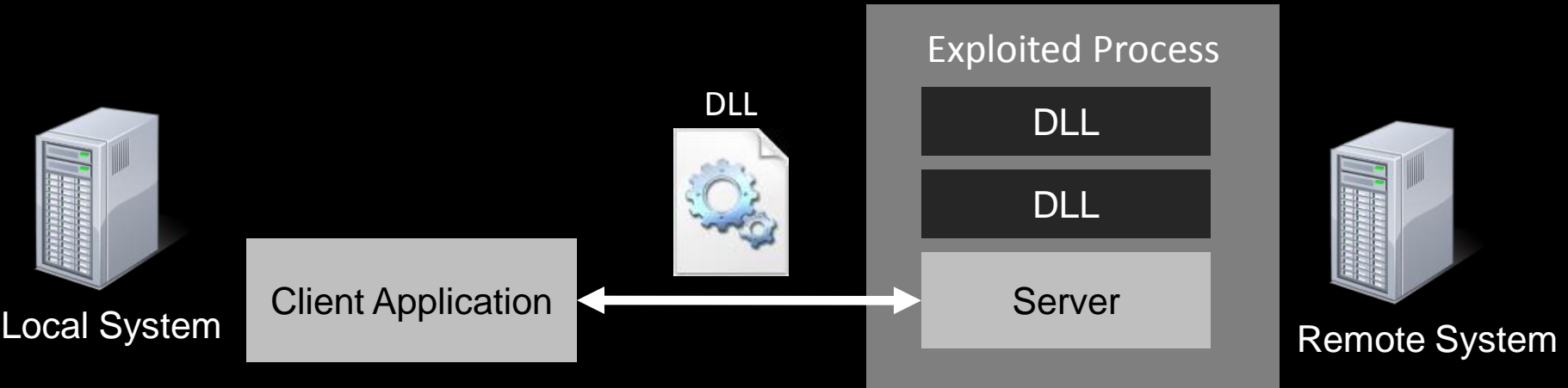
Compensates for general nature of the server



In-Memory Library Injection

$$-k_B \sum_i P_i \log_e(P_i)$$

Below Gotham
Laboratories



Example

Metasploit's Meta-Interpreter (Meterpreter)

Extensible remote shell that's delivered in an exploit payload

Extensions are implemented as DLLs rather than as raw machine code

Sam Juicer: a Meterpreter extension that dumps password hashes without disk writes

Implementation

Routines used by the dynamic loader are hooked

Enables a DLL's byte stream to be loaded from memory

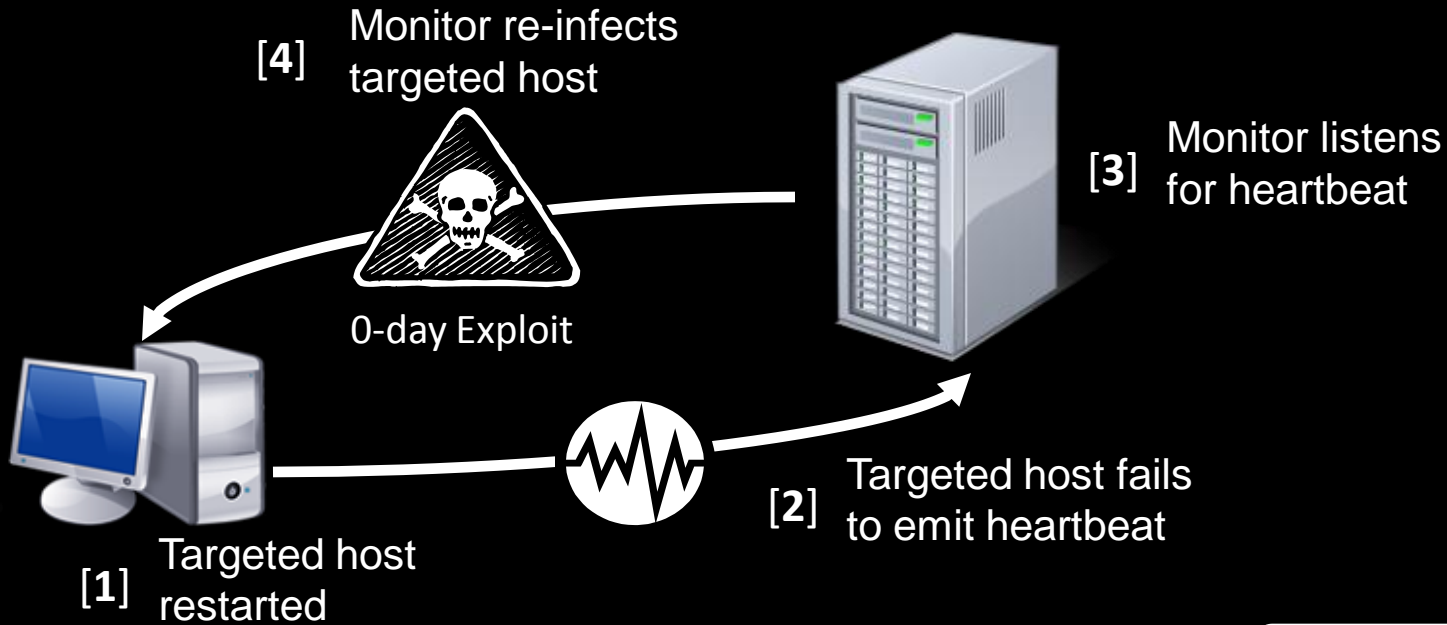
<http://www.nologin.org/Downloads/Papers/remote-library-injection.pdf>



Persist via Re-Infection

$$-k_B \sum_i P_i \log_e(P_i)$$

Below Gotham
Laboratories



Notes

Heartbeat could be a signal transmitted over a *passive* covert channel (PCC)
Don't generate any traffic of our own, merely alter existing packet streams

<http://invisiblethings.org/papers/passive-covert-channels-linux.pdf>

Based on ideas presented by Joanna Rutkowska

<http://www.blackhat.com/presentations/bh-federal-06/BH-Fed-06-Rutkowska/BH-Fed-06-Rutkowska-up.pdf>

$$-k_B \sum_i P_i \log_e(P_i)$$



Can also avoid the disk by hiding in firmware

Public Research

John Heasman, <http://www.blackhat.com/presentations/bh-federal-06/BH-Fed-06-Heasman.pdf>
Anibal Sacco and Alfredo Ortega, <http://cansecwest.com/csw09/csw09-sacco-ortega.pdf>
Darmawan Salihun, *BIOS Disassembly Ninjutsu Uncovered*, A-List Publishing, 2006

Commercial

Absolute Software sells Computrace, which includes a BIOS-based persistence agent
<http://developernet.absolute.com/products-core-technology.asp>
Several OEMs have embedded this agent at the firmware level
<http://www.absolute.com/partners/bios-compatibility>

Scenario

Someone figures out how to commandeer Computrace...



$$-k_B \sum_i P_i \log_e(P_i)$$

Introduction

The Quandary of Live Response
Another Option: Post-Mortem Analysis
Anti-Forensic Strategies

Tactics & Countermeasures

Forensic Duplication
Recovering Files
Recovering Deleted Files
Capturing a Metadata Snapshot
Identifying Known Files
File Signature Analysis
Static Analysis of an .EXE
Runtime Analysis of an .EXE

Data Source Elimination

Memory-Resident Rootkits
Firmware-Based Rootkits

Operational Issues

Footprint and Fault-Tolerance
Launching a Rootkit
Conclusions



$$-k_B \sum_i P_i \log_e(P_i)$$



The Tradeoff

Minimize Footprint → Sacrifice Restart Survival

May need to balance the two based on:

- The type of environment being targeted
- The value of the data to be acquired
- The skill level of your opponent(s)

Caveats

If the value of the data warrants the necessary R&D, you can have both

Periodic shutdowns can occur even in high-end environments

The Chicago Stock Exchange reboots its machines every evening

http://staging.glg.com/tourwindowsntserver/CHX/pdf/tech_road.pdf

$$-k_B \sum_i P_i \log_e(P_i)$$

Assuming a knowledgeable, well-armed, adversary...

Preferred Vector: Install a Memory-Resident Rootkit via an Exploit

Everything happens inside of an existing process (no need to launch a new one)
Can avoid disk modification entirely (though traces may reside in the page file)

Less Attractive Vector: Install an Agent in the Firmware

Firmware launches a bare-bones server that loads the rootkit proper over a socket
Leaves a minimal amount of code on the system, in a spot that's often ignored

Least Attractive Vector: Persist Somewhere on Disk

Initiating code will, by necessity, be naked and accessible
You can expect that your code will, with enough effort, be discovered
Leverage the five anti-forensic strategies with defense in depth to buy time



$$-k_B \sum_i P_i \log_e(P_i)$$

Bottom Line

State of the art anti-forensics can defeat disk analysis
(Perhaps this explains why this is a relatively inactive sub-field?)

Observation

A rootkit may never use disk storage...
But it still has to *execute in memory*
And it will almost always *talk to the outside*

Corollary

The arms race continues in the domains of live response and NSM



For More Information...

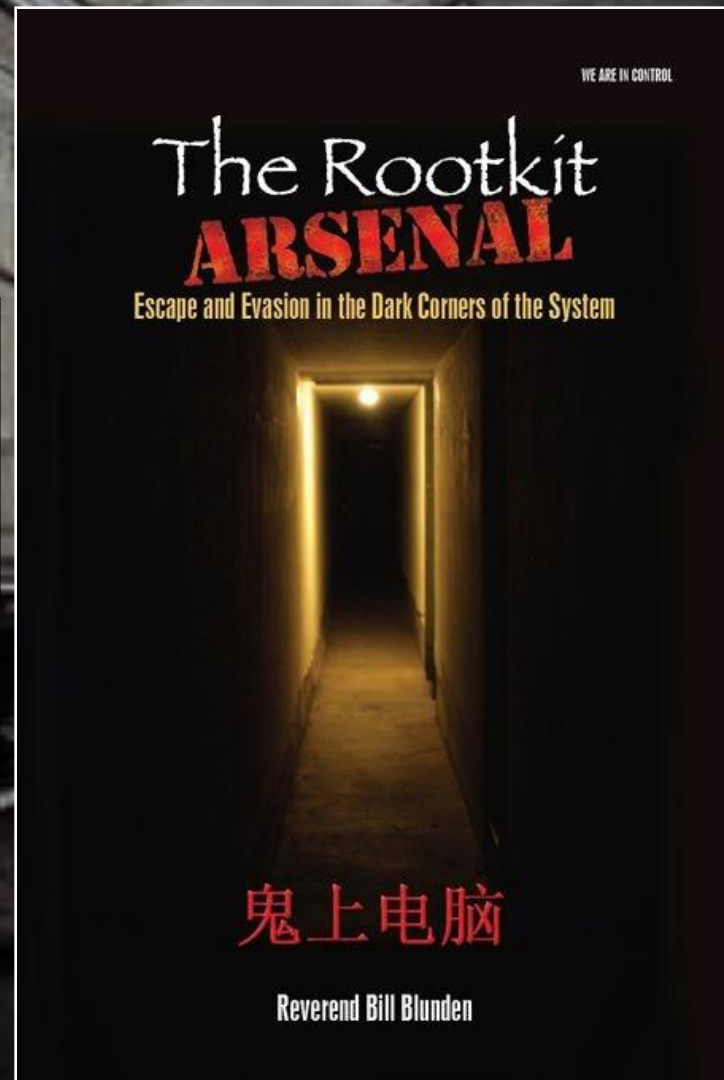
$$-k_B \sum_i P_i \log_e(P_i)$$

Below Gotham
Laboratories

The Rootkit Arsenal

Jones & Bartlett Publishers (May 4, 2009)

ISBN-10: 1598220616



Thanks and Greetings

$$-k_B \sum_i P_i \log_e(P_i)$$

**Below Gotham
Laboratories**

To Security Researchers Who Shared with the Rest of Us

David Aitel, John Ayccock, Richard Bejtlich, BigBoote, Darren Bilby, Maximiliano Cáceres, Brian Carrier, Jamie Butler, James Foster, Dawid Golunski, Glyn Gowing, the grugq, Nick Harbour, John Heasman, Greg Hoglund, Alex Keller, George Ledin, Elias Levy, The Linux-NTFS project, Vinnie Liu, Mark Ludwig, Mathew Monroe, Mental Driller, NV Labs, H D Moore, Metasploit, Jeff Moss, Gary Nebbett, Matt Pietrek, Pluf, Ripe, Marc Rogers, Mark Russinovich, Joanna Rutkowska, Darmawan Salihun, Bruce Schneier, Sherri Sparks, skape, Skywing, Sven Schreiber, Alexander Tereshkin, Irby Thompson, Jarkko Turkulainen, Dmitry Vostokov

Questions?

$$-k_B \sum_i P_i \log_e(P_i)$$

**Below Gotham
Laboratories**

Thank You for Your Time

