Digital Anti-Forensics:
Emerging trends in data transformation techniques

Christian S.J. Peron &
Michael Legary
Seccuris Labs

Abstract

This paper explores two questions: What methods can be used to deceive someone who is in an investigative role into trusting an object which has been exploited? What kind of impact does operating system and application run-time linking have on live investigations? After experimenting with dynamic object dependencies and kernel modules in the UNIX environment, it is the opinion of the authors that run-time linking can be exploited to alter the execution of otherwise trusted objects. This can be accomplished without having to modify the objects themselves. If an investigator trusts an inherently un-trusted object, it can result in the possible misdirection of a digital investigation.

Keywords: anti-forensics run-time linker hijack

1 Introduction

Judgments we make ultimately affect our decisions, which in turn affect our actions. In a similar sense, investigations are based on a set of judgments which are influenced by technical and non-technical factors. If those factors are in any way different from normalcy, a judgment will occur and ultimately affect the decisions of an investigator.

Criminals who intend to remain hidden try to take advantage of the factors which influence judgments made during preliminary investigations[1]. In regard to digital forensics, those factors which affect judgment tend to be the integrity of certain datasets which are commonly used to form evidence.

If an attacker manipulates the information in those datasets such that anomalies, failures, or specific outcomes do not arise, an investigator will have no grounds to suspect the integrity of the data. This could lead the investigation away from the source criminal activity.

This paper will review traditional methods used by adversaries mainly within UNIX environments to mislead investigators performing online or live analysis of systems. Additionally, less mainstream methods used to deceive investigators will be examined along with the manner in which they can be detected.

2 Anti-forensics

Over the past decade academic researchers, law enforcement agencies, and investigative practitioners have worked together to create processes around the evolving discipline of Digital Forensics. This discipline has been founded through traditional forensic study which combines science and technology to investigate and establish facts in courts of law. As standardized processes and techniques are established and adopted into the daily practices of forensic analysis the techniques utilized by adversaries will change in an attempt to overcome detection.

2.1 Anti-forensic methods

The primary building blocks for any type of digital evidence consist of electronic data. If an adversary can limit the identification, collection, collation and validation of electronic data the ability for an investigator to develop clear evidence will be hindered. To achieve this goal

[1] Preliminary investigations in this context refer to the initial suspicions that a system has been exploited which lead to online investigations or analysis.

an adversary can destroy, hide, manipulate or prevent the creation of evidence.

2.2 Destruction of data

The most basic set of techniques in anti-forensics consist of the physical and logical destruction of data or evidence. Physical destruction can be accomplished with the use of magnetic techniques such as the degaussing of media or through the application of brute force.

Logical destruction includes techniques that reinitialize media or significantly change the composition of data residing on the media[2]. The goal of physical and logical destruction techniques is to significantly damage integrity, or remove traces of relevant data from a targeted media. Although destruction of data can be the most thorough and efficient means to limit the amount of evidence an investigator can identify and collect, it also limits the ability of the adversary to achieve their motives.

2.3 Hiding of data

Obfuscation and encryption of data give an adversary the ability to limit identification and collection of evidence by investigators while allowing access and use to themselves. Obfuscation techniques include the use of Steganography[3] which allows the hiding of data in such a manner that the presence or contents of the data can not be identified. Encryption techniques may not prevent the detection of data, but will limit access to data contents by modifying data in such a manner that it will be unintelligible to investigators.

2.4 Data creation prevention

During the regular transactions between an adversary and an information system, a large amount of potential evidence is created. If an adversary can prevent the creation of relevant data before an investigator has the ability to identify or collect it, they can improve their chances of success in bypassing detection. Direct prevention techniques include the use of Root Kits[4] or the modification of system binaries such that the ability to generate relevant and credible data has been removed from the system. This method will limit the creation of data, but may also limit functionality of the system itself in turn limiting the ability of an adversary to achieve their goals.

2.5 Emerging techniques

In an attempt to maintain functionality of a system while limiting the creation of relevant data adversaries utilize data transformation techniques which maintain or re-establish an investigator's trust in a system. Transformation techniques are utilized by Root Kits or rogue shared libraries which can hijack system calls or run time linkages to modify data during the creation process. Data identified and collected by and investigator that was created during the transformation process will not be relevant to the investigation.

This paper will focus on and discuss data transformation techniques adversaries utilize to prevent the creation of data in an effort to limit an investigators ability to identify direct or circumstantial evidence throughout an investigation. Conventional and advanced methods that adversaries utilize will be outlined along with traditional detection techniques investigators use. Finally, emerging techniques that are uncommon in standard adversary modus operandi will be introduced along with possible

---

[2] A lot of secure deletion utilities operate by performing a series of operations like setting every bit of a file to 1 or true, then setting every bit to 0 or false.

[3] Steganography is the practise of hiding one piece of information inside another. An example might be putting a message inside a .JPG image file to bypass text based pattern matching.

[4] A root kit is generally a tool or collection of tools used on a compromised note to create back doors, capture or hide various types of information like passwords, network traffic or other resources.

detection strategies to identify the use of such techniques.

3 Conventional transformation methods

3.1 Initial system compromise

Processes or practices performed during investigations are generally in response to a security vulnerability which has been discovered, targeted, and exploited by an adversary. Most security-related incidents occur due to a lack of effective information technology governance and management within an organization. However, it should be noted that even with management direction and policy enforcement, break down in technical implementation can result in servers, workstations, and other network devices to remain un-patched. This problem is exponentially worse if management policies do not exist or are not enforced.

3.2 Deception of security personal

Once an adversary has compromised a component of a remote system, their primary goal is to retain control over the newly acquired resources for as long as possible. This is achieved through deceiving administrative and investigative individuals, ensuring that the adversary is not discovered.

An adversary is commonly discovered when the existence of certain unauthorized objects are verified. These objects can include: files, directories, processes, jobs, network connections, or listening services. Conventionally, these objects are hidden from regular operations through the modification or replacement of various system utilities made available in most UNIX environments.

Disk-related UNIX utilities such as df, ls, and du, are normally used for displaying mounted file systems, disk consumption, and searching or displaying information about file and file meta-data. Modification of these types of utilities may help hide files, file hierarchies, and disk consumption.

Process-related UNIX utilities, such as ps, top, crontab, and w, are used for listing information about processes, threads or Light Weight Processes, or jobs that have been scheduled to run. Modification of these types of utilities could result in processes or jobs being hidden.

Network-related UNIX utilities, such as netstat, sockstat, fstat, and tcpdump, were designed to list information about network connections and sockets[5]. When modified, these utilities could help to hide sockets, listening services, network connections, and general network activity existing on the system.

In addition, an adversary may introduce a backdoor or Trojan into a commonly used application or process. This provides a covert channel which enables an adversary to gain access to the system, bypassing any auditing or logging mechanisms put in place by the operating system.

4 Advanced transformation methods

4.1 OS kernel services

Most modern operating systems support multi-tasking or multi-programming environments. Each task or thread of execution is known as a process, which is uniquely identified by an integer-based Process ID (PID). Typically, a process executes in two different contexts: user-state and system or kernel-state. Manipulating data flow between these contexts can be instrumental to successfully hiding information from live investigation processes.

In kernel-space, operations like interrupt handling, credential management, operating system access control, and scheduler activations are performed. While the current execution context remains in kernel-space, manipulation of

[5] In general network sockets are treated the same as file handles. Except rather than the end point being a file, it can be another host or node on a network.

system control registers and the execution of privileged instructions is permitted.

The kernel is responsible for servicing hardware and software interrupts. Software interrupts, or "system calls", which are services or functions provided by the operating system. These software interrupts are a critical component in the day-to-day operations of any system. Among many other things software interrupts are responsible for: opening files, handling disk and network I/O, credential management, and modification of file attributes.

System calls are generally represented by integers. The value of the system call is its index inside a global array within the kernel. These arrays are also known as a software interrupt vectors. Most operating systems provide a facility which allows system administrators to load kernel modules. This enables for programmers to execute code in the context of the kernel and allows system administrator to dynamically add and remove functionality from a running system. This ability also helps operating system developers to write new parts of the kernel without constantly rebooting the system.

Generally, these system calls are responsible for performing access control functionality. This ensures that a subject has the appropriate access level to interact with an object through credential management, control of file attributes, permissions, and ownerships. The integrity of these system calls is paramount to the security of the entire system.

4.2 Kernel modules and hijacking system calls

By enabling the dynamic linker facility within the kernel, the administrator provides an adversary the facilities to load modules which can re-initialize the elements or entries stored within the software interrupt vectors. This allows an adversary to effectively "hijack" software interrupts. Essentially, an adversary can load a kernel module, re-initialize a system call stored within the interrupt vector to reference a malicious block of code contained within the adversary's module.

By hijacking software interrupts, an adversary can manipulate information which is returned to user-space utilities, such as ps, netstat, or ls. This allows an adversary to hide files, processes, network connections, and other objects which could be exposed through a software interrupt, without actually modifying the user-space utilities which could otherwise expose them.

4.3 Detection of system call hijacking

Some operating system kernels provide the facilities for dynamic module loading, and in some cases, provide a facility to prevent it. For instance, some operating systems create different security levels which make extremely difficult for an adversary to load dynamic functionality into the kernel. This can be accomplished by restricting access to certain system calls and writing to kernel memory devices like /dev/mem[6].

Detection of a malicious or rogue kernel module can be very difficult. Depending on how complex the kernel module itself is, it is possible that the adversary can hijack system calls responsible for the loading and unloading of kernel modules themselves. Otherwise, it may be possible to detect the presence of a kernel level "root-kit" which is hijacking software interrupts by comparing the address currently loaded into the slot against the address of the original system call.

Essentially, when the adversary loads their module and re-defines the system call, they change the address of the system to their own. Therefore, if the address for the system call, which is stored in the interrupt vector, is different than the address of the proper system

[6] /dev/mem is a pseudo device which provides an interface to the physical memory of the system. Reading or writing to /dev/mem is the same as reading or writing the physical memory of the system.

call, it can be determined within a good degree of certainty that the system call has been hijacked (Figure 1).

```
/*--
 * In FreeBSD the operating system, “interrupt vectors” for system calls
 * are stored as arrays called “sysent arrays”.  The sysent data structure
 * contains a member called “sy_call” which stores the address of the
 * routine to execute when a software interrupt comes in.
 *
 * It is important to note that an operating system can have many different
 * interrupt vectors. For example if the operating system supports many
 * emulations, typically each emulation will have it’s own interrupt
 * vector table.
 */
if (sysent[SYS_open].sy_call != open)
        panic(“open system call has been hi-jacked”);

if (sysent[SYS_write].sy_call != write)
        panic(“write system call has been hi-jacked”);
```

Figure 1:
Code snippet for the FreeBSD [1] *operating system which when executed in the context of the kernel, could be used to detect the presence of a hi-jacked system call.*

5 Traditional transformation detection methods

In order to detect possible intrusion events, system administrators and investigators look for network and system anomalies, modification of system objects, and unusual system resource consumption.

5.1 Cryptographic hashing for data integrity

Cryptographic hashing implements concepts provided by mathematical one-way functions to verify the integrity of data. Using these methods, files can be uniquely identified through fingerprints. File fingerprints are generally calculated using cryptographic hashing functions like MD5 [2], SHA [3], and RIPE-MD [4].

The goal of these functions is to input some message or file of arbitrary length, and compute a fingerprint of constant length. Hash functions are also known as “one way functions”, meaning they should be trivial to calculate in one direction, but computationally infeasible to calculate in the other[7]. Changing a single bit within the file should result in a drastically different fingerprint.

Most host-based intrusion detection systems (HIDS) or root-kit detection agents operate on the same premise. To verify the authenticity of a file, these systems calculate the fingerprint or digest of the original file in its trusted state, and store the fingerprint in a trusted location. A fingerprint is calculated on the current file, and compared against the trusted fingerprint. If the two fingerprints conflict, it is safe to assume the current file has been modified (Figure 2).

```
% md5 ps.trusted
MD5 (ps.trusted) =
9501ef286ef3ab8687b7920ca4fee29f
% md5 /bin/ps
MD5 (/bin/ps) =
02b2f8087896314bafd4e9f3e00b35fb
%
```

Figure 2:
Output of the “md5” utility. Hashes for the “ps.trusted” file and “bin/ps” file are calculated based on the MD5 cryptographic hashing algorithm and compared

[7] It is possible for two different files to result in the same checksum or hash. This is known as a hash collision. The chances that two file hashes can collide can be calculated using the something called the “Birthday Paradox” [5]

5.2 Process analysis

Processes can be thought of as threads of execution which are represented by data structures in memory. The entities contain objects like: open files (file descriptor table), memory maps or VM objects, ownership labels, and resource consumption statistics. In most UNIX environments, details about sockets, network connections and open file handles are normally located in a descriptor table stored within a process object.

There are several utilities that analyze information about these process data structures. These utilities can be used to locate various objects that an adversary would like to keep hidden.

The process objects can be retrieved through different portals including /dev/mem, an interface to the systems physical memory, and procfs, which implements a view of the system process table and makes it accessible inside a special file system.

In some cases, the utilities themselves could be modified to not report information on certain processes. In this case, the use of cryptographic checksums can be used along with cross referencing output with information stored in the process file system can help to identify the source of any discrepancies.

5.3 Signature/pattern matching

Many adversaries use pre-written "root kits" to aide them in hiding or covering up objects they do not want investigators to find. If these root kits are made available to the public, samples or specific patterns can be extracted and used to discover the potential presence of the same kit on other nodes. (Figure 3)

Detection agents typically contain a database of commonly known patterns and signatures. Using this database, detection agents perform routine scans of system memory and files, iterating through signatures looking for matches.

```
% file libtransform.so.1
libtransform.so.1: ELF 32-bit LSB shared object, Intel 80386, version 1 (FreeBSD), stripped
%
```

Figure 3:
Output of the "file" utility on a shared object. The "file" utility attempts to figure the file type for a specified file. This is accomplished by performing a number of tests, including one which checks for particular fixed data formats. I.E. ELF [6] *executable file formats*

5.4 Network monitoring

In some cases, intrusion events are detected by various network monitoring mechanisms. These mechanisms can include: network intrusion detection systems (NIDS), firewall monitoring, and bandwidth utilization trending. Many NIDS base their detection through the use of pattern matching and signatures. Sensors monitor network traffic looking for traces of known attack signatures and generate alerts based on positive matches.

Signatures can take on a number of forms. In many cases, signatures which are found in most common attacks are matched against binary sequences stored in network transmissions. (Figure 4) However, when monitoring transmissions that are unencrypted, a network intrusion detection sensor can monitor for the output of utilities which require super user access. For example, the output of the id(1) command returning UID 0.

```
Nov 10 21:59:06 <4.1> 172.16.1.20 snort: [1:466:1] SHELLCODE x86 stealth NOOP [Priority: 2]:
{PROTO001} 10.0.1.125 -> 10.5.1.3
```

Figure 4:

This is an example Snort [7] i*ntrusion detection log which has detected the op-codes or machine instructions for a "stealth NOOP" (machine instruction). This log contains information about the transmission source and destination IP addresses and services in which the pattern was recognized.*

On compromised hosts, processes can be created which listen on network sockets for instructions from the adversary, similar to a Trojan. These instructions can tell a process to launch a packet flood against a specific target. When the host initiates an attack against the target, it commonly generates a bandwidth utilization anomaly which can be detected through traffic trending. In some cases adversaries can expose themselves by connecting to services which are not permitted by firewall devices. (Figure 5)

```
% tcpdump -nett -i pflog0
listening on pflog0, link-type PFLOG (OpenBSD pflog file), capture size 96 bytes
1100221136.677441 rule 1/0(match): block in on sis0: IP 10.0.0.35.4646 > 205.11.11.11.445: S
552159036:552159036(0) win 64240 <mss 1460,nop,nop,sackOK>
1100221138.370423 rule 1/0(match): block in on sis0: IP 10.0.0.35.4646 > 205.11.11.11.445: S
552159036:552159036(0) win 64240 <mss 1460,nop,nop,sackOK>
```

Figure 5:

This is an example of some packets which have been analyzed using the tcpdump [8] *program on the OpenBSD* [9] *PF Firewall after the packets have been logged via the pflog virtual interface.*

6 Emerging transformation methods

6.1 Program execution

A new process is initially created when its parent process forks its execution using the fork system call. Once the resources and scheduler activations have been allocated and registered for the process object itself, another system call, "exec", gets called. The exec system call will replace the current execution image with the executable image associated with the new program.

These executable images or machine instructions are loaded into memory, where the kernel pre-emptively runs the task or machine code on a time-shared basis. In short, programs can be thought of as a set of functions which is executed in some logical order which is determined at run-time.

Routines or functions in a computer program can be thought of as entry points to a series of instructions located in computer memory. Calling function "X" is equivalent to jumping to a particular position in memory and executing the code located there. These entry points have an attribute called a "symbol", which uniquely identifies the routine. Processes maintain symbol tables which allow the kernel to locate instructions based on symbol name.

6.2 Run-time linker

Programs on a system often share the same functions. As the number of utilities or programs increase, the concept of building a shared library where all programs can share functions, rather then replicating the same code, started to surface. Most UNIX variants have a "Run-Time Linker", which provides this functionality. Commonly used functions, such as printf[8] and fgets[9], are stored in a share object. Trends

[8] printf is standard function defined in ISO/IEC (``ISO C90'') to print formatted data to the screen.

[9] fgets is a standard function defined in ISO/IEC (``ISO C90'') to read a line from a stream.

indicate that more and more operating systems are moving towards implementing dynamically linked programs by default (Table 1).

As the program executes with the requirement to execute a particular shared function, the pages associated with that function will be mapped into the process' address space from the shared object using the mmap[10] system call. This process is also known as "on demand paging".

Operating System	Dynamically Linked Fundamental Utils.
Solaris 10 [10]	YES
Redhat Linux 9 [11]	YES
FreeBSD 5.x	YES
FreeBSD 4.x	NO
OpenBSD 3.5	NO
NetBSD 1.6.X [12]	NO
NetBSD 2.0	YES
Apple's Darwin 7.3.1 [13]	YES

Table 1:
This table displays which operating systems ship with dynamic or run-time linking on fundamental system binaries by default. These binaries are typically found in the /bin and /sbin directories.

6.3 Preloading shared objects

The run-time linkers can be configured for a variety of different types of real-time linking operations. In some cases, the user may choose to avoid "on demand paging" and have the functions mapped into the address space immediately, improving run-time performance.

In other cases, the user may choose to preload another shared library before any other library gets loaded. This allows programmers to override certain functions that the application will execute, without actually modifying the application itself.

6.4 Hijacking of user space library calls

Depending on the level of privilege acquired by an adversary on a compromised machine, they could be in a position where they can manipulate run-time linking operations of otherwise trusted processes. *This allows an adversary to change the execution behaviour of a process, without actually modifying the binary or kernel interrupt vectors.* This implies that investigators relying on cryptographic checksums to verify the integrity of programs execution, could be mislead or have a false sense of trust in the binary in question.

The following example will illustrate how an adversary might mislead investigators by utilizing the run-time linker, manipulating certain system logs which contain incriminating information without modifying system binaries, operating system interrupt vectors, or the log files themselves.

The adversary writes a shared library which defines two entry points or symbols: recvfrom[11] and write[12]. These functions are utilized by processes like syslogd, (a daemon which centrally manages logs used by most processes on the system) to read and write messages to/from a logging or network sockets.

These newly defined entry points have an interesting characteristic: they are capable of performing transformations on the data before submitting it to the syslog process. The module opens a "recipe" file and loads in a series of "transformation instructions". These

[10] mmap is a function which maps files or devices into memory. This interface is not standard, but most operating systems implement it.

[11] recvfrom is an optional system call provided by most UNIX variants to receive messages from a socket.

[12] write is a standard system call defined by POSIX.1 for writing information to a descriptor. Descriptors may include handles to files or sockets.

transformation instructions determine what words or patterns are to be substituted with adversary-defined data.

This allows the adversary to change, delete, or transform system logs centrally, without actually editing the log files themselves. Even if the system has write operations to the file system being audited, the modifications to the log files will be dispatched as the syslogd process.

In a common case scenario, the system might have a host-based intrusion detection agent running, which monitors commonly used system binaries for any kind of modification, ranging from content, permission, or attribute modifications. However, the adversary does not have to modify the binaries of the syslogd or any of its usual shared objects in order to change the execution behaviour of the process.

One technique the adversary may use is to replace any occurrence of their IP address or subnet with an arbitrary address, or delete any log which contains their IP address or subnet pattern all together. Commonly, logging operations could be tested and, in most cases, logs generated by investigators would be processed and un-touched by the adversary's transformation rules.

Since the adversary may have applied these transformation filters to any outgoing and incoming logs, network-based intrusion detection sensors or firewall devices that are logging to the compromised device could also be modified. If the syslog server has been configured to log to a remote host, transformations will be applied to those messages as well. This could result in an external trusted logging server housing incorrect or modified logs.

It should be noted that this sort of attack can occur in several different ways: An adversary could implement these type of hooks in other commonly used functions, such as writes to STDOUT[13] via the printf function (which would manipulate the information returned from system utilities like ps, netstat or fstat), or writes to independent log files from other processes. Any executable files which depend on run-time linking can be affected by these types of attacks.

7 Emerging transformation detection methods

7.1 Shared library analysis

When extensive analysis on processes is being performed, it is recommended that a vmcore[14] be captured and transferred to a trusted system. Utilities, such as lsof [14], can be used to analyze the process data structures within the vmcore. Any irregular shared library which has been opened by the process and mapped into the process's address space should be an indicator to the investigator that something suspicious is occurring

[13] STDOUT is a standard output stream defined by ISO/IEC (``ISO C90") which is typically used for writing conventional output

[14] A virtual memory core (vmcore) is a snapshot of the current memory. On some operating systems VM cores can be created by using the "halt" command. These cores can be used by utilities like ps to retrieve the current process state of the OS at the time the core was created.

.

```
Dev0# lsof -p 40374
COMMAND   PID USER   FD   TYPE     DEVICE SIZE/OFF    NODE NAME
test    40374 root  cwd   VDIR      233,9      512 5888013 /usr/home/modulus/transform
test    40374 root  rtd   VDIR      233,4     1536       2 /
test    40374 root  txt   VREG      233,9     4795 5888022 /usr/home/modulus/transform/test
test    40374 root  txt   VREG      233,4   136276     553 /libexec/ld-elf.so.1
test   40374   root      txt          VREG                  233,9               7130   5888027
/usr/home/modulus/transform/libtransform.so.1
test    40374 root  txt   VREG      233,4   869100     550 /lib/libc.so.6
test    40374 root    0u  VCHR        5,1  0t46554      93 /dev/ttyp1
test    40374 root    1u  VCHR        5,1  0t46554      93 /dev/ttyp1
test    40374 root    2u  VCHR        5,1  0t46554      93 /dev/ttyp1
test    40374 root    5u  PIPE 0xc190e000    16384         ->0xc190e0ac
dev0#
```

Figure 6:

The "lsof" utility will retrieve process objects from a system core and process the file descriptor tables so an analyst can investigate any activities which require the use of open files.

This process contains an open file handle to an object called "libtransform.so.1". Further analysis indicates that alternate dynamic symbols or functions are being registered for the recvfrom and write entry points.

```
dev0# objdump -T libtransform.so.1 | grep text
0000083c l    d  .text  00000000
00000b94 g    DF .text  0000009a recvfrom
00000c30 g    DF .text  00000099 _write
dev0#
```

Figure 7:

Information from the objdump [15] *utility is being passed into a grep command to filter out any irrelevant information. Objdump is responsible for displaying information about object files, and can be used to analyze the details about specific aspects of the object files, such as what dynamic symbols it contains. Shared libraries are essentially special object files.*

The results of objdump illustrate that alternate entry points have been registered for the recvfrom and write library functions. This may indicate that the adversary has hijacked these functions to manipulate the operating environment to avoid detection. Any information which was obtained or produced as a result of these calls can not be trusted.

In many operating systems, the pre-loading of shared objects is accomplished through the use of process environment variables. Less complex implementations of this attack may be detectable by analyzing the process's environment variables. It should be noted that this should be a less trusted technique because an adversary is able to re-define environment variables within their shared object.

8 Conclusions

An adversary can utilize run-time linking mechanisms to alter the execution logic of either an operating system or application without having to modify the binary. Commonly, integrity checks are based on the ability to detect modifications to the binary itself. Objects which appear to be trust-worthy, possibly are not. This can result in compromised logging, audit or information sources being trusted by investigative bodies, resulting in the possible the avoidance of more thorough offline forensic analysis or the misdirection of the investigative bodies themselves.

9 References

[1] Project FreeBSD. FreeBSD Home Page. http://www.FreeBSD.org. November, 1993.

[2] Rivest R. The MD5 Message-Digest Algorithm, IETF RFC 1321. ftp://ftp.rfc-editor.org/in-notes/rfc1321.txt. April, 1992.

[3] Burrows J. The Secure Hash Standard, FIPS PUB 180-1, IETF RFC 3174. http://csrc.nist.gov/cryptval/shs.html. 1993.

[4] Dobbertin Hans, Bosselaers Antoon, Preneel Bart. The RIPEMD-160 Cryptographic Hash Function. http://www.esat.kuleuven.ac.be/~bosselae/ripemd160.html. 1988-1992.

[5] Eric W. Weisstein. "Birthday Attack." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/BirthdayAttack.html

[6] Unix System Laboratories. The Executable and Linking Format, http://x86.ddj.com/ftp/manuals/tools/elf.pdf.

[7] Project Snort, The Snort Home Page. http://www.snort.org. 2002.

[8] Project Tcpdump, The Tcpdump Home Page. http://www.tcpdump.org. 2002

[9] Project OpenBSD. The OpenBSD Home Page. http://www.OpenBSD.org. 1996.

[10] Sun Microsystems. The Solaris Operating System, SunOS. http://www.sun.com. 1989.

[11] Red Hat Inc. The Redhat Linux Operating System. http://www.redhat.com. 1993.

[12] Project NetBSD. The NetBSD Home Page. http://www.netbsd.org. 1993

[13] Apple Inc. The Darwin Operating System. http://www.apple.com, http://www.apple.com/opensource/. 2001.

[14] Abell V. The lsof (LiSt Open Files) Project. http://people.freebsd.org/~abe/.

[15] GNU Binutils. The Binutils Home Page. http://www.gnu.org/software/binutils/.