Software Engineering Institute

The MAL: A Malware Analysis Lexicon

David A. Mundie
David M. McIntire

February 2013

TECHNICAL NOTE
CMU/SEI-2013-TN-010

CERT® Program

http://www.sei.cmu.edu

Carnegie Mellon

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM-0000216

Table of Contents

Acknowledgments **v**

Abstract **vii**

1 Introduction **1**

2 Development of the MAL **3**

2.1 Origins of the MAL 3

2.2 Building the MAL 3

2.3 Publishing the MAL 5

3 Future Research **7**

3.1 The MAL V2 7

3.2 Taxonomic and Ontological Analysis 7

3.3 Community-Wide Deployment 7

3.4 Malware Thesaurus 7

3.5 Improved Hosting 7

3.6 Competency-Based Analysis 7

4 The Malware Analysis Lexicon **8**

References **33**

List of Figures

Figure 1: Preliminary MAL Mind Map 6

Acknowledgments

The authors would like to thank John Ackley for providing the email archive for our corpus, Alex Nicoll for his generous and insightful suggestions, and Paul Ruggiero for his patient editing.

Abstract

The lack of a controlled vocabulary for malware analysis is a symptom of the field's immaturity and an impediment to its growth. Malware analysis is a splintered discipline, with many small teams that for cultural reasons do not, or cannot, readily communicate among themselves; this condition encourages the growth of many local dialects. This report presents the results of the Malware Analysis Lexicon (MAL) initiative, a small project to develop the discipline's first common vocabulary.

1 Introduction

In 2011, the U.S. Department of Defense asked the JASON program to "examine the theory and practice of cyber-security, and evaluate whether there are underlying fundamental principles that would make it possible to adopt a more scientific approach" [MITRE 2010].

The first JASON program report concluded that

> *The most important attributes would be the construction of a common language and a set of basic concepts about which the security community can develop a shared understanding… a common language and agreed-upon experimental protocols will facilitate the testing of hypotheses and validation of concepts.* [MITRE 2010]

In short, the JASON program found that cybersecurity is a relatively immature discipline that has still not reached agreement on its foundations. To use an analogy from resilience management modeling folklore [Caralli 2011], it is as though a team of doctors were asked to remove a patient's gall bladder without agreeing on what a gall bladder is.

Nowhere in the cybersecurity community is this lack of a common vocabulary, and the problems it causes, more apparent than in malware analysis. This emerging discipline is rapidly growing in importance. The demand for malware analysis teams is increasing so quickly that hiring, training, and retaining staff are becoming major problems for many organizations. A recent proposal for a Defense Advanced Research Projects Agency (DARPA) program to improve the training of malware analysts noted that in the case of a "doomsday scenario" in which a large-scale cyber-warfare attack targeted the United States, the ability to quickly ramp up the nation's supply of malware analysts would be a critical success factor for cyber-defense. Yet malware analysts are currently trained largely by apprenticeship methods that do not scale well to internet time. The results are problematic:

1. Malware analysis teams and their customers misunderstand each other. Anecdotes abound: To take one example, a customer will ask to know everything about a piece of malware, and the analysts will provide an expensive, time-consuming reverse engineering analysis, when in fact a simple superficial analysis would have sufficed.
2. Hiring is hit-or-miss. The jobs that analysts perform are so poorly understood they are often referred to as a "black art." To an outsider, it looks as though analysts just stare at indecipherable codes on a screen for hours and hours, and then suddenly understand how the malware works. The opacity of their work makes it difficult to make intelligent hiring decisions.
3. Progress toward a malware analysis body of knowledge is impeded. As the JASON quote above implies, and as we will argue later in this report, a controlled vocabulary is the first step in turning ad hoc folklore into a science, and no controlled vocabulary has been developed for malware analysis.
4. Certification programs for malware analysts are problematic because there is no standardized way to assess the abilities of a malware analyst.

5. The malware analysis community remains balkanized, with many isolated teams and only limited data sharing. The need to preserve competitive advantage compounds the disincentives to share data.

This report presents the results of the Malware Analysis Lexicon (MAL) initiative, a small project that addresses the discipline's lack of a common vocabulary.

2 Development of the MAL

2.1 Origins of the MAL

Carnegie Mellon University's Software Engineering Institute (SEI) has a long history of raising the maturity level of computer-intensive disciplines: The Capability Maturity Model (CMM®), the CERT® Resilience Management Model (CERT®-RMM), and the Team Software Process℠ (TSP) are a few examples. Each of those efforts began by defining its terms.

The CERT Program within the SEI has played a similar role for information security organizations over the last 15 years and has helped dozens of information security teams from around the world stand up or sustain their operations. In particular, the CERT Program's CSIRT (Computer Security Incident Response Team) Development Team (CDT) has studied the state of the art in incident management, provided courses in creating and managing information security teams, and produced methodologies for assessing incident response teams.

Most recently the CDT has been developing a body of knowledge for incident management and has published a preliminary description of the process [Mundie 2012]. As part of that project, the authors examined a wide range of extant bodies of knowledge and, based on that examination, developed a systematic, 10-step process for constructing a body of knowledge. Perhaps unsurprisingly, the process starts with the production of a controlled vocabulary to standardize the terms to be used. On that foundation, the process constructs categories, taxonomies, and a variety of ontologies (static, dynamic, and intentional).

The current project was inspired by that work. We view the MAL as applying the first step in that 10-step process to the domain of malware analysis. We were further inspired by traditional lexicography as embodied in *The Oxford Guide to Practical Lexicography* [Atkins 2008] and by the repository of controlled vocabularies on the *Controlled Vocabularies* website [Riecks 2012].

2.2 Building the MAL

The lack of a controlled vocabulary for malware analysis is both a contributing factor and a symptom of the immaturity of the field. Malware analysis is a splintered discipline, with many small teams that for cultural reasons do not, or cannot, readily communicate among themselves; this condition encourages the growth of many local dialects.

The language of the malware analysis community is, according to one experienced malware analyst, derived in equal parts from computer science, criminology, and geek culture [Gennari 2011]. It is a jargon that is in constant flux, paralleling the evolution of the internet and information technology, as well as the results coming out of the malware research community. There is little standardization in this jargon, even on common terms such as "zero day" or "runtime analysis."

® CERT is a registered mark owned by Carnegie Mellon University.

Field Research

Our original intention was to collect vocabulary items using techniques from ethnography or industrial-organizational psychology, that is, by interacting with and observing the daily activities of the CERT malware analysis team. We selected that approach in part because we think it is the best way to build a larger malware analysis body of knowledge. As industrial ethnographers never tire of pointing out, the only way to accurately find out what people do is to watch them do it.

Unfortunately the ethnographic approach proved to be too time consuming and disruptive for the malware analysis team. Instead we turned to another technique, which is in many ways more interesting from a lexicographic perspective: Text mining the last 10 years' worth of email from the CERT malware analysis team.

First, we tokenized the email archive and removed duplicate tokens, leaving approximately 90,000 terms. Next, we ran the tokenized list through some filters developed by the CERT vulnerability team to remove nontechnical words, on the grounds that syncategorematic terms and terms denoting common nontechnical items, such as "lunch," are unlikely to have developed special meanings in the domain of malware analysis. Such semantic extension is not impossible, but it is unlikely enough to be ignored. The filters remove any words that occur in *Moby Dick* and a couple of other corpora; unfortunately the filters do not perform lemmatization, so they did not eliminate forms of words that occur in *Moby Dick* only with different inflections.

The filtered list still contained about 70,000 terms. Analyzing them manually was unappealing, so we resorted to another heuristic. We reasoned that most of the important terms for the MAL would be fairly frequent in the email corpus. As with semantic extension, an extremely important term could have appeared only once or twice during the 10 years of correspondence, but we were willing to accept the possibility that such words would be excluded.

Zipf's law [Weisstein 2012] was our friend. Of the 70,000 terms, half only occurred once or twice, and only about 5,000 occurred more than five times. We filtered those 5,000 manually to identify the most likely candidates, leaving a list of about 700 terms. We gave them to an experienced malware analyst, who picked out about 40 of them for inclusion in the lexicon.

In her book *Lost in the Web of Words* [Murray 1977], K.M. Elisabeth Murray recounts the biography of her grandfather, James Murray, and his heroic, Victorian project to build the first edition of the *Oxford English Dictionary*. He recruited a network of lexicographers throughout England who sent him newly observed senses of words on index cards, which he laboriously sorted by hand in wooden pigeonholes. Comparing our efforts to his makes it clear how far computerization has advanced lexicography. We think our efforts illustrate the potential of text mining for accumulating corpora, but also the necessity of resorting occasionally to unsatisfying heuristics and the method's relatively low yield, in our case just 41 terms out of 90,000 source tokens.

Textbooks

The second source of items for our corpus was textbooks. Relatively few books are dedicated to malware analysis and reverse engineering. We chose three of them, *The Shellcoder's Handbook: Discovering and Exploiting Security Holes* [Anley 2007], *Reversing: Secrets of Reverse*

Engineering [Eilam 2005], and *The Malware Analyst's Cookbook* [Ligh 2010], along with the workbook for the CERT course on malware analysis [SEI 2012].

Our process was straightforward. We scanned for likely terms in the books' indexes. When necessary we would consult the text of the books for definitions, look them up online, or consult an expert. This yielded about 200 terms that arguably represent a solid starting point for developing a shared malware analysis vocabulary.

Internet Resources

The third source of lexical items for the MAL was the internet. Given the closed nature of the malware analysis community, we were surprised to discover a wealth of terms discussed in the open source online literature. With only minimal searching, we uncovered about 25 new terms.

The authority of those terms must be considered carefully and is undoubtedly less than that of textbooks and the email corpus, which have both been vetted by experienced experts. Nonetheless, we feel that further exploration of the open source literature is a promising avenue for enlarging the corpus.

2.3 Publishing the MAL

Early in the project, we made the decision to store our lexical entries in MultiMarkdown [Fletcher 2012], a dialect of John Gruber's Markdown language that supports definition lists. Some form of XML would have worked, but the ease of editing Markdown won out. We also considered Microsoft Word and Excel, as well as mind mapping, but the simplicity of Markdown, and the added interest of evaluating a technology that was new to us, spurred our use of MultiMarkdown.

In general we have been happy with the result. The definition lists consist of terms on their own lines, separated from their definitions by a colon as the first character in the definition line. This is even easier to process in scripts than XML is, and it facilitates editing.

In terms of distributing the MAL, we envisage three distribution channels in addition to this technical note.

DICT Server

First, our vision has always been to stand up an aggregated SEI dictionary server that can be used to look up all the terms from all the dictionaries and glossaries that the SEI has published, and to add the MAL to that server. A proof-of-concept server running DICT [Faith 1997], the Internet Engineering Task Force's (IETF's) standard for dictionary servers, has been implemented on a test network as part of this project. It runs the GNU implementation of DICT and contains the following dictionaries:

- the 250-word MAL dictionary
- a 580-word insider threat glossary
- a 2,000-word general information-security dictionary
- a 260-word CERT-RMM glossary
- a 230-word CMM dictionary

ePub

Second, we created an electronic book version of the MAL. We feel that the advantages of this format are well worth the small incremental cost. The distribution of the MAL in this format is under development.

Mind Map

Third, we converted the MAL to an interactive mind map based on a preliminary classification of the terms into categories such as "traditional computer security terms" and "anti-reversing terms." It provides a convenient interface for browsing and editing the MAL and is a first step toward a full malware analysis ontology.

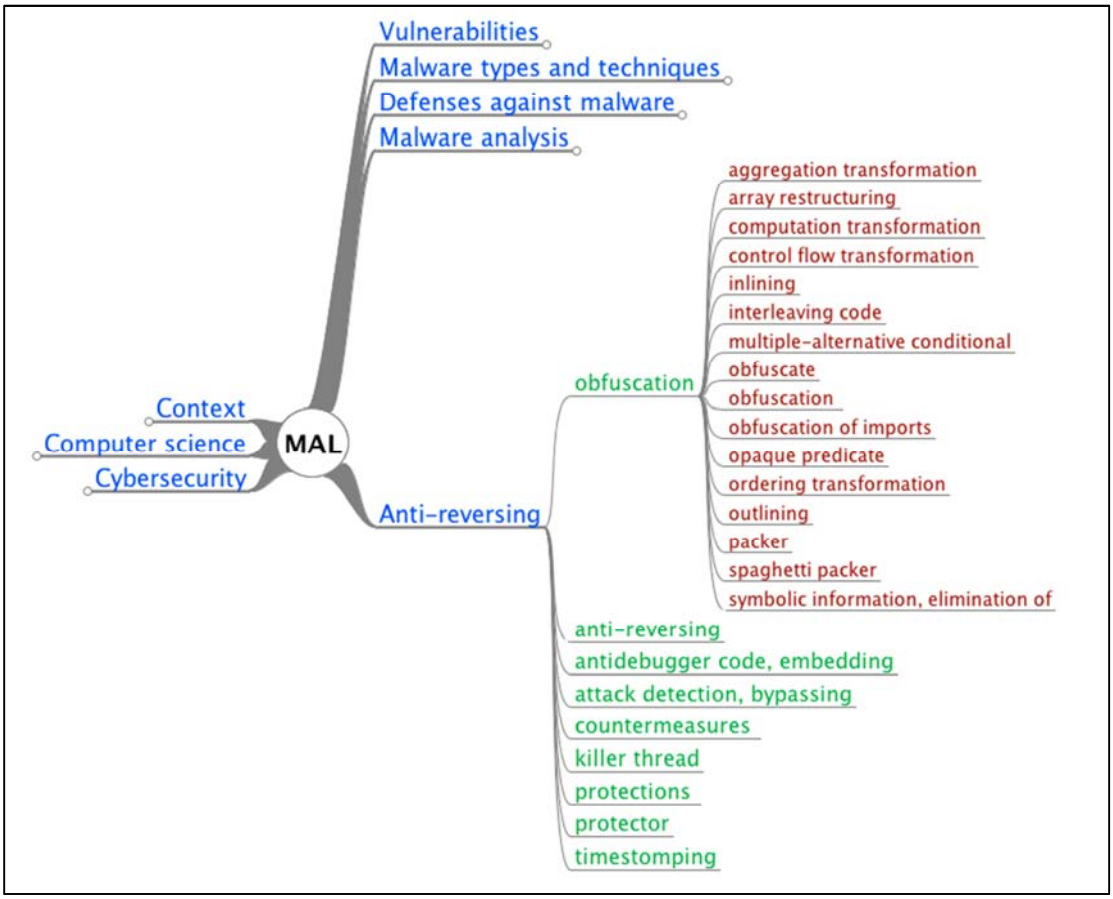

Figure 1: Preliminary MAL Mind Map

3 Future Research

The current project is a small pilot that has, in our opinion, demonstrated the feasibility and utility of a malware analysis lexicon. We would like to build on this work in a variety of ways.

3.1 The MAL V2

The coverage of the lexicon could be expanded, especially with respect to internet resources. A modest additional effort could probably increase the number of terms by 50 percent.

3.2 Taxonomic and Ontological Analysis

A controlled vocabulary is the first step in the methodology for building bodies of knowledge [Mundie 2012]. The second step is to analyze the terms in the vocabulary, looking for hierarchies and similarities, to create a categorization for the domain of interest. The third step is to construct an actual ontology that captures relationships among the terms. The last two of these steps would be relatively straightforward for the MAL. The mind-map version of the MAL incorporates an initial categorization of the controlled vocabulary and could be the basis for further development.

3.3 Community-Wide Deployment

To have an impact on the malware analysis community, the MAL needs to grow in two dimensions. First, it needs to be vetted by a broader public of malware analysts. The definitions included in the MAL seem reasonable to us, but they must be reviewed and improved by end users. Secondly, it needs to be publicized and marketed to achieve buy-in by the community at large.

3.4 Malware Thesaurus

The lexicon as it stands now is a traditional, flat dictionary. We feel that moving to a richer format such as WordNet or WordVis would greatly improve the usability of the lexicon.

3.5 Improved Hosting

The DICT server in its current state has limitations that are not acceptable beyond the pilot. For example, it does not support partial-word searches. A very small effort could remove these limitations and provide a much more usable interface to the MAL.

3.6 Competency-Based Analysis

The MAL was originally one component of a proposed larger study of how malware analysts perform their jobs. Now that we have the lexicon, we feel it would be beneficial to carry out the rest of that study in order to understand exactly what is involved in the many facets of malware analysis. This would yield benefits in many areas such as negotiating analysis work and hiring, training, and managing analysts.

4 The Malware Analysis Lexicon

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

0–9

0-day vulnerability
See *zero-day vulnerability*.

A

AAAS
See *ASCII armored address space*.

address space layout randomization (ASLR)
An anti-malware security method that involves randomly arranging the positions of key data areas, usually including the base of the executable and the position of libraries, the heap, and the stack, in a process's address space, making it more difficult for an attacker to predict target addresses.

advertising-supported software
See *adware*.

adware
Any software package that automatically renders advertisements. Frequently contains spyware such as keyloggers.[1]

aggregation transformation
A control-flow transformation that attempts to break the abstractions created by the programmer so that the high-level organization of the code becomes senseless.[2]

alphanumeric filters, defeating
A malware technique that constructs malware using a limited character set so as to be acceptable to the software being attacked. Often used for buffer overflow attacks.

alternate encoding
A non-ASCII, non-Unicode encoding, often used to bypass web filters. *Examples:* hexadecimal, octal, or URL encoding.

anti-debugger code, embedding
An anti-reversing technique that causes the program being dynamically analyzed to perform operations that would damage or disable a debugger.[2]

anti-reverse-engineering clause
A clause in a software license agreement that prohibits reverse engineering of the licensed product. The Digital Millennium Copyright Act (DMCA) is an anti-reverse-engineering law that applies to products protected by digital rights management (DRM). In Europe, decompilation is permissible for interoperability purposes.

anti-reversing
A set of tools and techniques that helps developers harden their application code against reverse engineering.[3]

appender
A file-infecting virus that places its code at the end of the files it infects, adjusting the file's entry point to cause its code to be executed before that of the original file.[4]

application layer attack
An attack that exploits a vulnerability at the application level of the Open Systems Interconnection (OSI) model.

architectural failure
A vulnerability caused by faults in overall system design.

array restructuring
An anti-reversing technique that obfuscates arrays by methods such as concatenating multiple arrays into one, interleaving multiple arrays, splitting one array into multiple smaller ones, or changing the dimensionality of an array.

artifact
Any file created by an exploit and remaining on a compromised machine after the exploit. Most often used to refer to malicious code.

ASCII armored address space (AAAS)
An anti-malware protection mechanism that protects ("armors") executable code by adding ASCII characters that foil malicious code. *Example:* Prepending a zero byte (indicating end of C string) to stack data so that string-writing routines terminate before overwriting the stack.

ASCII Venetian implementation
An exploit writer that uses the Venetian technique. See *Venetian technique*.

ASLR
See *address space layout randomization*.

attack detection bypassing
A collection of techniques that defeat attempts to discover malicious actions at runtime. *Example:* alternate encodings.

auditing source code
The practice of examining source code to ensure it has not been tampered with or modified in an unauthorized fashion.

authentication token
See *cryptographic token*.

authorization problem
A problem that results from insufficient or incorrect permissions and that complicates analysis.

automated malware analysis
Use of systems or programs that automate a portion of the malware analysis process. Includes but is not limited to unpacking, static analysis, and runtime analysis.

B

backdoor access
Use of a hidden access mechanism previously inserted in a program or system.

behavior blocker
A piece of software that monitors a system for suspicious actions and blocks the software that is executing them. Suspect actions may include editing the system registry, writing to protected system areas, or harvesting email addresses or password data.[5]

behavioral analysis
See *dynamic analysis*.

binary auditing
The practice of auditing compiled code to ensure that the binaries in question have not been altered.

binary registry pattern matching
The practice of searching the binary of an artifact for particular patterns indicative of certain properties.[6]

bit flipping
A technique used in fuzzers that randomly inverts a single bit in input strings, in the hopes of triggering a crash and thus revealing an exploitable security bug.

blacklist
A list of hashes or other objective indicators of files known (or suspected) to be malicious.

blended threat
A sophisticated attack using multiple malware types and vectors to penetrate and control a system.[7]

bochs
An open source processor emulator for x86 and x86–64 processors.

boot sector virus
A virus that infects the master boot record of a storage device.[8]

boundary problem
An exploit caused by input that crosses a trust boundary. *Example:* SQL Injections.[9]

bridge building
A malware technique used to defeat alphanumeric filters by using opcodes with alphanumeric bytes to write the real shellcode.

browser helper object
An object inserted into a web browser that can sometimes have malicious functions.

buffer overflow
A vulnerability or means of exploiting a vulnerability in a piece of software designed to have data passed into it. In software with this vulnerability, the area set up to hold this data (the buffer) is inadequately defined and controlled. By passing the right amount of data in, an attacker can overflow the buffer and place data of their choice, such as code to disable defenses or download further malicious code, into an area where it will be executed by the target system.[10]

bug discovery and exploitation
The practice of discovering flaws in software and exploiting them for malicious purposes.

C

cavity filler
A type of file-infecting virus that seeks out unused space within the files it infects and inserts its code into these gaps to avoid changing the size of the file and thus hiding its presence from integrity-checking software.[11]

class break
A single attack that can be used to break a significant number of similar devices.[12]

cleanware
Euphemism for a computer program with no malicious functionality; however, so-called cleanware programs are commonly used maliciously.

cloning
Making a copy of a piece of cleanware, frequently to inject malware into the copy.

code injection
A class of attacks that exploit a lack of proper data validation to insert unintended code into an application, for example by means of cookies, URLs, or SQL commands.[13]

code interleaving
An error-correction technique that guards against bursts of errors by spacing the codeword symbols out through interleaving.[14]

code-level reversing
The practice of reverse engineering an artifact to the point where a functionally similar version of it has been reconstructed with new code.

code patcher
A tool to facilitate byte-level modifications to object code without recompilation or reassembly.[6]

Common Type System (CTS)
An Ecma International standard that specifies how type definitions and specific values of types are represented in computer memory. It is intended to allow programs written in different programming languages to easily share information.[15]

companion virus
A virus that takes the place of a particular file on a system instead of injecting code into it.[16]

computation transformation
1. A control flow transformation that reduces the readability of the code by modifying the program's original control flow structure to make a functionally equivalent program that is far more difficult to translate back into a high-level language, either by removing control flow information or by adding new control flow statements that complicate the program and cannot be easily translated into a high-level language.[2]
2. An obfuscation technique that makes code less readable by transforming its topology while preserving its semantics.
3. An obfuscation technique that inserts new (redundant or dead) code to hide the real control flow behind irrelevant statements.[17]

configuration changing
The alteration of a system or network's software, settings, or hardware.

control flow statement
A statement that affects the flow of the program based on certain values and conditions.[2]

control flow transformation
A transformation that alters the order and flow of a program in a way that reduces its human readability. Subdivided into computation transformations, aggregation transformations, and ordering transformations.

countermeasures
1. Techniques used by malicious programs to prevent them from being analyzed inside virtual machines.
2. Techniques used by malware analysts to prevent malicious programs from harming their systems.

cracking
The "dark art" of defeating, bypassing, or eliminating any kind of copy protection scheme.[2]

crackme
A program written to provide an intellectual challenge to crackers and to teach cracking basics to "newbies."[2]

cross-platform shellcode
An exploit that works on multiple instruction set architectures by finding instructions that act as jumps in one architecture and no-ops in the others.[18]

cryptographic token
1. A physical device that an authorized user of computer services uses to implement two-factor authentication.
2. A software token. See also *software token*.
3. A USB token.
4. An authentication token.

crypto-processor
1. A microprocessor that can directly execute encrypted code by decrypting it on the fly.[2]
2. A processor designed specifically to execute cryptographic algorithms faster than a general-purpose processor.

cryptor
A packer that encrypts the code it packs, either only on disk or on disk and in memory.[19] See also *packer*.

CTS
See *Common Type System*.

cwsandbox
A commercial runtime sandbox.

D

data collection
In tracing for vulnerabilities, capturing and managing the information generated by function hooks. See also *tracing for vulnerabilities*.

data diddler
A type of malware that makes small, random changes to data, such as data in a spreadsheet, to render the data contained in a document inaccurate and in some cases worthless.[20]

data reverse engineering
The process of deciphering program data such as undocumented file formats and network protocols, commonly for the purpose of achieving interoperability.[2]

decompilation
The creation of a high-level language list that accurately reflects the semantics of a compiled computer program (often using a disassembly as input).

decompiler
A program used to break down the compiled code of another program into the commands that make up that code, in a higher level language than disassembler output. The data can then be used as part of the process of reverse engineering to analyze the code and its intentions.[21]

deobfuscator
A program that uses data-flow analysis algorithms to remove irrelevant instructions from an obfuscated program and restore its original structure.[2]

design recovery
A subarea of reverse engineering in which domain knowledge, external information, and reasoning are added to the observations of the subject system to identify meaningful higher level abstractions beyond those obtained directly by examining the system itself.[22]

disassembling
The creation of an assembly language list that accurately reflects the semantics of a compiled computer program.

dropper
See *dropper file.*

dropper file
A type of Trojan that deposits an enclosed payload onto a destination host computer by loading itself into memory, extracting the malicious payload, and then writing it to the file system.[23]

dynamic analysis
The study of how malware behaves when it is executed: what gets installed, how it runs, and what it communicates with.[24]

dynamic patching
Changing the code while the target executable is loaded into memory.

E

eHash
See *exact hash*.

EICAR test file
A small, simple file intended to be used to verify that anti-malware software is functioning correctly.[25]

entry point
Location in a compiled computer program where execution is to begin. See also *original entry point*.

exact hash
A cryptographic hash of a sequence of unmodified bytes. See also *position-independent code hash*.

exploitative malware analysis
An advanced form of malware analysis that demonstrates thorough understanding of the malicious code by modifying it or using it for unintended purposes.

extinct bug class
A type of vulnerability that was commonly found in open source software previously, but that is no longer found in modern versions of that software. *Example:* `strcpy` and other string manipulation functions in C.

F

fastflux
A technique for rapidly mutating the IP addresses to which an internet domain name resolves.

fault injection
A technique for improving the coverage of a software test by introducing faults to test code paths, particularly error-handling code paths, that might otherwise rarely be followed. Types of fault injection include robustness testing, syntax testing, and fuzz testing.[26,18]

file-backed section object
A section object that is attached to a physical file on the hard drive.[2]

file infector virus
A virus that infects a system by inserting itself somewhere in existing files; this is the “classic” form of virus.[27]

fingerprinting
A technique for identifying malware by examining characteristics such as opcode frequencies, API call vectors, or flow-graph structural properties.

fork bomb
A process that repeatedly makes copies of itself using the `fork()` system call. The intent of a fork bomb is to use all possible entries in the system process table to ensure no other processes can be started, creating a denial of service on the machine.[28]

forward engineering
The traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system.[22]

function hooking
A technique used in runtime vulnerability discovery that substitutes hooks in a function table so as to instrument functions of interest.

function-level working-set tuning
Working-set tuning based on function-level reorganization.[2]

fuzz testing
See *fuzzing*.

fuzzer
A tool that performs fuzzing.

fuzzing
A form of fault injection commonly used to test for security problems in software or computer systems by providing invalid, unexpected, or random data to the inputs of a computer program and monitoring the program for exceptional behavior.[29]

G

generic detection
Recognizing malware by its similarity to known items.[30]

generic logic error
A nonspecific vulnerability due to misuse of data structures or classes. Frequently the root cause of a security issue.[18]

greyware
Software that, while not definitely malicious, has a suspicious or potentially unwanted aspect.[31]

ground rule for reversing sessions
A general methodology applied to each artifact analysis task by a reverse engineer.

H

hash-and-decrypt
A mesh design pattern for protection mechanisms. At each stage of the encryption, a hash function is used to derive a key that is then used to decrypt the next stage.[32]

heap protection
A technique designed to detect heap overflows after the fact, for example by checking pointer consistency when calling `unlink`.[33]

Hex-Rays
A commercial decompiler.

hooking
A method of capturing calls to external libraries or system functions using techniques such as injecting code into known function entry points, modifying binary header information, or overriding interrupt handlers.

I

IDA Pro
A commercial disassembler.

idb
The database file saved by the Hex-Rays IDA Pro disassembler tool.

idc
A scripting language for the Hex-Rays IDA Pro disassembler software.

imaging
The practice of capturing a bit-for-bit copy of any digital media.

implant
Object code inserted into an existing program using a code patcher or other tool.[6]

infector
A function of malware that alters target files for the purpose of persisting and hiding the injected malware.

information-stealing worm
A worm that encrypts information assets on compromised systems so they can only be decrypted by the worm's author (for purposes of blackmail, for example).[2]

information theft
The unauthorized access or removal of data.

initial classification
See *initial triage*.

initial triage
The process by which artifacts are sorted upon receipt and analysis plans are created.

inlining
An anti-reversing technique that uses the compiler optimization technique of expanding function calls in place (i.e., inlining) to confuse the analyst about function boundaries.[2]

instrumented dynamic analysis
The analysis of malware by executing it within a debugger.

interleaving code
A form of obfuscation that splits code into sections that are rearranged and connected by unconditional jumps.[2]

IPS
Intrusion prevention system.

K

keygenning
The process of creating programs that mimic the key-generation algorithm within a protection technology to provide an unlimited number of valid keys.[2]

keylogger
Malware that records key presses.[34]

killer thread
An anti-reversing technique that makes malware analysis more difficult by killing the malicious program if it seems to be running within a debugger.

kleptographic worm
See *information-stealing worm*.

L

lazy binding
See *lazy linking*.

lazy linking
A linking strategy that performs symbol resolution when needed during program execution, not during execution startup.[18]

legality of reverse engineering
A debate over the situations in which reverse engineering should be permitted by law, usually revolving around the question of what social and economic impact reverse engineering has on society as a whole.[2]

linear sweep disassembler
A disassembler that processes instructions sequentially, in the order they appear in the code file. See also *recursive traversal disassembler*.[2]

M

machine-code analysis
Analysis performed on binary code without the corresponding source code.

macro virus
Virus that uses a macro language, for example in Microsoft Office documents.[35]

MAEC
See *Malware Analysis Enumeration and Characterization*.

malcode
Short for malicious code, also known as malware.

mallab
A common moniker for an air-gapped laboratory for the analysis of malware.

malware analysis attack vector
Any of the techniques malware analysts use to defeat malware: static analysis, dynamic analysis, static patching, dynamic patching, and dumping.[19]

Malware Analysis Enumeration and Characterization (MAEC)
A data exchange format and protocol for metadata and analysis of malware.

malware handling procedure
A documented set of processes used for safely handling malicious artifacts.

Malware Technical Exchange Meeting (MTEM)
An annual, classified conference on malware topics.

man-in-the-middle attack
1. An attack that utilizes flaws in a communication protocol that allow the attacker to become the broker of information between a victim process and the intended data recipient.
2. A form of data theft, carried out by the attacker being positioned between the victim and the intended destination for the data.[36]
3. A form of active eavesdropping in which the attacker makes independent connections with multiple victims and relays messages between them, making them believe that they are talking directly to each other over a private connection, when in fact the entire conversation is controlled by the attacker.[37]

masm
Microsoft Assembler, a commonly used assembler language representation.

McCabe software complexity metric
A measurement of linearly independent paths through a program's source code.

memory mapped file
A segment of virtual memory that has been assigned a direct bit-for-bit correlation with some portion of a file or file-like resource.

metamorphic virus
malware that changes its own code with each infection.[38] See also *polymorphism*.

metamorphism
The ability of an artifact to alter itself to a degree that detection mechanisms are defeated.

Metasploit
A public site of malware techniques, examples, and tools.

methodology of reversing
Any process used to reverse engineer one or more artifacts.

mid-infector
Malware that inserts its code in the middle of files.[39]

misinfection
A file infected incorrectly by a virus.[40]

mobile code
1. Code received from remote, possibly untrusted systems, but executed on a local system.[41]
2. Software transferred between systems (e.g., across a network) and executed on a local system without explicit installation or execution by the recipient.[42]

Examples: scripts, web applets, dynamic email, macros.

MSVC
Microsoft Visual C compiler and development suite.

MTEM
See *Malware Technical Exchange Meeting*.

multipartite virus
Malware that infects boot records, boot sectors, and files.[43]

multiple-alternative conditional
In reverse engineering, a control structure consisting of a series of if-then-else statements with each condition leading to a different code block. Sometimes used for obfuscation.[2]

N

Nagel algorithm
A congestion control technique that combines multiple small outgoing messages and sends them all at once to improve the efficiency of TCP/IP networks.

named objects
1. A technique used to share objects between different processes in the Windows environment.[44]
2. A compilation technique that ensures objects have unique names so access to them can be shared without confusion.[2]

National Software Reference Library
An initiative of the U.S. National Institute of Justice to collect digital signatures of known, traceable software applications to facilitate data collection on computer systems that have been seized in criminal investigations.[45]

ngram
A reference to N consecutive bytes; for example, a 32-bit word is a 4-gram.

NOP ramp
See *NOP slide*.

NOP sled
See *NOP slide*.

NOP slide
A malware technique that inserts extra no-op instructions in code to make it robust in the face of jump addresses that are only approximate.[46]

NSRL
See *National Software Reference Library*.

NVRAM invalidation
An attack against Cisco routers that causes a write protection error by writing to write-protected NVRAM, then reconfiguring the router when it reboots and requests configuration information.[18]

O

obfuscate
To make something difficult to understand or analyze.

obfuscation
A generic term for a number of anti-reversing techniques that reduce a program's vulnerability to any kind of static analysis by modifying the program's layout, logic, data, or organization to make it functionally identical but far less readable.[2]

obfuscation (methods)
Techniques used by malicious code to obfuscate itself.

obfuscation of imports
An anti-reversing technique that creates a nonstandard import table or otherwise hides system calls in an effort to confuse malware analysts.[6]

OEP
See *original entry point*.

on-access scanning
In anti-virus technology, scanning files for malware when they are accessed.[47]

one-factor exploit
An exploit that relies on knowing only one fact about its environment, such as the return address or the current program counter.[18]

opaque predicate
An anti-reversing technique that attempts to confuse recursive traversal disassemblers by inserting into code a false branch that appears conditional but is essentially unconditional.[2]

ordering transformation
A control flow transformation that attempts to randomize the order of operations in a program as much as possible. Considered less powerful than computation transformations or aggregation transformations.

original entry point (OEP)
Entry point of the unpacked original computer program. Packing introduces new code that performs the unpacking at runtime, which becomes the new entry point.

outlining
An anti-reversing technique that confuses reversers by creating unneeded functions for arbitrary sequences of inline code. Opposite of inlining.[2]

out-of-scope memory usage vulnerability
A vulnerability caused by the possibility of using memory regions outside their valid scope and lifetime.[2]

P

packer
1. Software that compresses other software.
2. A malicious tool that compresses and obfuscates software in order to defeat anti-reversing.[19]
See also *cryptor*.

partial attack
An attack that discloses a portion of memory but does not allow the attacker to specify which portion of memory.[48]

pefile
An open source tool for working with Portable Executable format files.

PEiD
A commercial tool for analyzing malware. PEiD inspects programs to determine whether they have been packed by one of the popular executable packers or copy protection products.[2]

phash
See *position-independent code hash*.

PIC
See *position-independent code*.

PIC hash
See *position-independent code hash*.

polymorphism
The ability of malicious code to change its signature to avoid detection.

position-dependent code
Object code that must be loaded into, and run from, a particular address in memory, because not all branches are relative to the program counter. Opposite of position-independent code.[49]

position-independent code (PIC)
Object code that can be executed regardless of its location in memory because all branches are relative to the program counter. Commonly used for shared libraries. Opposite of position-dependent code.[49]

position-independent code hash (PIC hash)
The cryptographic hash of a sequence of executable bytes where bytes that are believed (by heuristics) to be absolute addresses are converted to "don't care" values (usually zero), removing the location-specific information from the bytes before hashing.

power usage analysis attack
An attack that extracts information from chips by monitoring their power usage, for example extracting a private key by observing slight variations in power consumption of a decryption chip.[2]

prepender
Malware that inserts it code at the start of files.[50]

process attribute
Information about a process stored in the process table, usually grouped into identifying information, state information, and control information.

process injection
A method used to inject malicious executables into another process.

process monitoring
Act of actively checking processes currently running inside a particular computing environment.

propagator
A function of malware that introduces itself or other malware onto other hosts by various means.

protections
1. Methods by which malicious artifacts attempt to stop analysis from being performed on them.
2. Methods by which an analysis environment is protected from malicious code present in that environment.

protector
An advanced form of packer that, in addition to packing the code, uses code consistency checks, anti-debugging techniques, license verification, and other techniques to estimate the security of the environment.

Q

qemu
An open source processor emulator.

R

ransomware
A type of malware that encrypts files on a victim's system, demanding payment of a ransom in return for the access codes required to unlock the files.[51]

raw socket
An internet socket that allows direct sending and receiving of network packets that are not subject to the operating system's network stack handlers and verification mechanisms and that typically have not had the headers removed.

reclamation
See *reengineering*.[22]

recursive traversal disassembler
A disassembler that follows the control flow statements in the program and as a result is more tolerant of anti-disassembly tricks.[2] See also *linear sweep disassembler*.

redocumentation
A subarea of reverse engineering that creates or revises a semantically equivalent representation of a piece of software at the same relative level of abstraction, usually used as alternate views for a human audience.[22] *Examples:* dataflow diagram, control flow diagram.

redundancy elimination
A compiler optimization technique that searches for and eliminates duplicated code. Of little interest to reversers.[2]

reengineering
The examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form.[22]

regedit
Tool for viewing and modifying the Windows registry.[52]

relative virtual address (RVA)
An offset from the address of a module loaded in memory.[53]

relocatable code
Object code that requires special processing by a link editor or program loader to make it suitable for execution at a given location.[54]

renovation
See *reengineering*.[22]

restructuring
The transformation from one representation form to another at the same relative level of abstraction while preserving the subject system's external behavior.[22]

reverse engineering
1. The process of developing a set of specifications for a complex hardware system by an orderly examination of specimens of that system.[22]
2. The part of the software maintenance process that helps clarify the system so an analyst can make appropriate changes.[22]

reverser
One who practices reverse engineering.

reversing
See *reverse engineering*.

ripping
Extracting data from physical media, like CDs or magnetic tape.

ripping key-generation algorithms
A cracking technique that identifies the key-generation portion of a crackme and ports it into a stand-alone program.[2]

rogue anti-malware
Fake security product demanding money to clean phony infections.[55]

runtime analysis
A form of malware analysis that executes the malware in a controlled environment and observes its behavior and impact.[56]

runtime image patching
See *runtime patching*.

runtime patching
Applying a modification to a program while the program is in operation.

RVA
See *relative virtual address*.

S

sample bias
Analysis problem caused by sampling procedures that are insufficiently random.

sandbox
An isolated area of a program or operating system that is prevented from interacting with the operating system or the parent program.

SandNet
A network operating system that integrates standard communication protocols with industry- and manufacturer-specific control protocols to improve its connectivity, processing, monitoring, and management capabilities.[57]

sea monkey data
Data that expands when submitted to a program, as when quote characters are expanded to HTML character entities.[18]

section object
In Windows, a special chunk of memory that is managed by the operating system.[2]

security identifier (SID)
1. A unique, immutable identifier of a user, user group, or other principle who needs secure access.[58]
2. A unique, often immutable, identifier of a user or other principle who is governed by an access control mechanism.

security token
See *cryptographic token*.

shellcode
1. A small piece of code that activates a command-line interface to a system that can be used to disable security measures, open a backdoor, or download further malicious code.[59]
2. A small piece of code that opens a system up for exploitation, sometimes but not necessarily involving a command-line shell.

SID
See *security identifier*.

software maintenance
The post-delivery modification of a software product to correct faults, improve performance or other attributes, or adapt the product to a changed environment.[22]

software token
A security token stored on a general-purpose electronic device such as a desktop computer, laptop, PDA, or mobile phone, rather than on a dedicated hardware device. See also *cryptographic token*.

source code auditing methodology
Any documented method used to protect and ensure the integrity of source code.

spaghetti packer
A packer that obfuscates programs by emitting "spaghetti" code with a complex and tangled control structure.

sploit
Short for "exploit."

ssdeep
One of several open source fuzzy hashing tools.[60]

stack data protection scheme
Any mechanism that attempts to protect a stack from exploitation. *Example:* Stack-Smashing Protector, also known as SSP or ProPolice.

static patching
Changing the code located in the file of the executable.[19]

string filter
A string transformation that can make string-based exploits more difficult. *Example:* Incoming strings may be converted to Unicode, so that a malicious input would need to be an ASCII sequence that produces malicious Unicode output.[2]

symbolic information, elimination of
An anti-reversing technique that hinders reverse engineering by removing or renaming textual information in the program.[2]

Sysinternals
A collection of system administration tools from Microsoft. Commonly used by malware analysts.

system-level reversing
The process of reverse-engineering an artifact so an analyst can understand all its functions on a system.

T

table interpretation
An obfuscation technique that breaks a code sequence into multiple short chunks controlled by an unintuitive conditional code sequence.[2]

this pointer
In object-oriented programming, a pointer that references a specific instance of the class to which the current method belongs. Passed as an implicit parameter.[18]

thunk
A subdivision of a computer program function.

timestomping
An anti-reversing technique that creates incorrect timestamps to confuse malware analysts.

tracing for vulnerabilities
A form of vulnerability discovery that executes a piece of code that has been instrumented to detect vulnerabilities.

traffic decoder
A Network Forensic Analysis Tool (NFAT) that separates network traffic by protocol (email, web, etc.) to facilitate analysis.

Trojan horse
A piece of malicious code disguised as something inert or benign.

trusted computing
A generic term for secure platforms that incorporate a combination of hardware and software changes, including cryptographic engine chips, to make the devices tamper-proof.[2]

tuning working sets
An optimization technique that involves re-ordering code sequences to minimize load time. For reversers, it can provide hints about the most frequently used routines, but can also make code less understandable.[2]

typosquatting
The registration of domain names similar to those of popular websites.[61]

U

unsafe unlinking
An exploit technique that overwrites the back and forward headers in a linked list with values that pass checks for unsafe unlinking but still produce undesired results when the node is removed from the list.[18]

USB token
See *cryptographic token*.

V

Venetian technique
A method of writing Unicode-type exploits using only ASCII letters and numbers.

vgrep
A tool provided by Virus Bulletin to cross-reference the names given to malware by different products.[62]

virus
1. A self-replicating malicious program that requires human interaction to replicate.
2. A self-replicating program that runs and spreads by modifying other programs or files.[63]

Virustotal
A public, commercial site that detects malware in uploaded software.

vtable
A virtual function table. Vtables pose a problem for auditing binaries because tracking down the destination of a call on a virtual function can be difficult without runtime analysis.

vulnerability
An exploitable weakness in a computer system.

W

wabbit
A form of self-replicating malware that makes copies of itself on the local system. Unlike worms, wabbits do not attempt to spread across networks.[64]

working-set tuning
The process of rearranging the layout of code in an executable by gathering the most frequently used code areas in the beginning of the module to reduce memory consumption and program startup time.[2]

worm
1. A self-replicating malicious program that replicates using a network and does not require human interaction.[2]
2. A self-replicating, self-propagating, self-contained program that uses networking mechanisms to spread itself.[63]

W^X

1. A protection mechanism that makes writable memory nonexecutable and executable memory nonwritable (hence "W^X", W xor X).[18]
2. A memory protection feature of the OpenBSD operating system in which every page in a process's address space is either writable or executable, but not both.[65]

Y

YARA

See *Yet Another Regex Analyzer*.

Yet Another Regex Analyzer (YARA)

An open source malware identification and classification tool.[66]

Z

zero-day exploit

An attack that exploits a zero-day vulnerability.

zero-day kernel vulnerability

A zero-day vulnerability in kernel code. *Example:* The Duqu Trojan (possibly a son of Stuxnet) used a dropper file with a Microsoft Windows zero-day kernel exploit.

zero-day vulnerability

A vulnerability that has not been disclosed to the general public and so can be exploited before patches are available.

zip bomb

A file compressed into some archive format and that expands to an enormous size when uncompressed, often by looping over the extraction code until the system's resources are exhausted.[67]

---

1 http://en.wikipedia.org/wiki/Adware

2 Eilam, Eldad. *Reversing: Secrets of Reverse Engineering*. Wiley, 2005.

3 Ishaq, A.F.M. "Anti-reversing as a Tool to Protect Intellectual Property," 248-252. *Second International Conference on Engineering Systems Management and Its Applications*. Sharjah, United Arab Emirates, Mar. 2010. IEEE 2010.

4 http://www.virusbtn.com/resources/glossary/appender.xml

5 http://www.virusbtn.com/resources/glossary/behaviour_blocker.xml

6 Software Engineering Institute. *Malware Analysis Apprenticeship*. Software Engineering Institute, Carnegie Mellon University, 2012. http://www.sei.cmu.edu/training/P88.cfm

7 http://www.virusbtn.com/resources/glossary/blended_threat.xml

[8] http://www.virusbtn.com/resources/glossary/boot_sector_virus.xml

[9] http://en.wikipedia.org/wiki/Fuzz_testing

[10] http://www.virusbtn.com/resources/glossary/buffer_overflow.xml

[11] http://www.virusbtn.com/resources/glossary/cavity_filler.xml

[12] ARM Security Technology. *Building a Secure System Using TrustZone® Technology*, Glossary-2. ARM Limited, 2009. http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf

[13] http://www.owasp.com/index.php/Code_Injection

[14] Borda, Monica. *Fundamentals in Information Theory and Coding*, 340. Springer-Verlag, 2011.

[15] http://en.wikipedia.org/wiki/Common_Type_System

[16] http://www.virusbtn.com/resources/glossary/companion_virus.xml

[17] Preda, Mila Dalla. *Code Obfuscation and Malware Detection by Abstract Interpretation*. Università degli Studi di Verona, 2007.

[18] Anley, Chris, et al. *The Shellcoder's Handbook: Discovering and Exploiting Security Holes*. Wiley, 2007.

[19] Lyashko, Alexey. "Dynamic Code Encryption as an Anti Dump and Anti Reverse Engineering Measure." *System Programming*, Mar. 2, 2012. http://syprog.blogspot.com/2012/03/dynamic-code-encryption-as-anti-dump.html

[20] http://www.virusbtn.com/resources/glossary/data_diddler.xml

[21] http://www.virusbtn.com/resources/glossary/decompiler.xml

[22] Chikofsky, Elliot J. & Cross, James H., II. "Reverse Engineering and Design Recovery: A Taxonomy." *IEEE Software 7*, 1 (January 1990): 13-17.

[23] http://www.symantec.com/security_response/writeup.jsp?docid=2002-082718-3007-99

[24] Distler, Dennis. *Malware Analysis: An Introduction*. SANS Institute, 2007.

[25] http://www.virusbtn.com/resources/glossary/eicar_test_file.xml

[26] http://en.wikipedia.org/wiki/Fuzz_testing

[27] http://www.virusbtn.com/resources/glossary/file_infector_virus.xml

[28] http://www.virusbtn.com/resources/glossary/fork_bomb.xml

[29] http://en.wikipedia.org/wiki/Fuzz_testing

[30] http://www.virusbtn.com/resources/glossary/generic_detection.xml

[31] http://www.virusbtn.com/resources/glossary/greyware.xml

[32] Lawson, Nate. "Mesh Design Pattern: Hash-and-Decrypt." *root labs rdist*, April 9, 2007. http://rdist.root.org/2007/04/09/mesh-design-pattern-hash-and-decrypt/

[33] http://en.wikipedia.org/wiki/Heap_overflow

[34] http://www.virusbtn.com/resources/glossary/keylogger.xml

[35] http://www.virusbtn.com/resources/glossary/macro_virus.xml

[36] http://www.virusbtn.com/resources/glossary/maninthemiddle_attack.xml

[37] http://en.wikipedia.org/wiki/Man_in_the_middle_attack

[38] http://www.virusbtn.com/resources/glossary/metamorphic_virus.xml

[39] http://www.virusbtn.com/resources/glossary/midinfector.xml

[40] http://www.virusbtn.com/resources/glossary/misinfection.xml

[41] Brown, Lawrie. *Mobile Code Security*. School of Computer Science, Australian Defence Force Academy, 1996.

[42] http://en.wikipedia.org/wiki/Mobile_code

[43] http://www.virusbtn.com/resources/glossary/multipartite_virus.xml

[44] Microsoft. "Object Names." *Windows Dev Center – Hardware*. Microsoft, 2012. http://msdn.microsoft.com/en-us/library/windows/hardware/ff557762%28v=vs.85%29.aspx

[45] National Institute of Standards and Technology (NIST). *National Software Reference Library*. NIST, 2012. http://www.nsrl.nist.gov/

[46] http://en.wikipedia.org/wiki/NOP_slide

[47] http://www.virusbtn.com/resources/glossary/onaccess_scanning.xml

[48] Parker, Timothy. *Protecting Cryptographic Keys and Functions from Malware Attacks*. University of Texas at San Antonio, 2010.

[49] http://en.wikipedia.org/wiki/Position-independent_code

[50] http://www.virusbtn.com/resources/glossary/prepender.xml

[51] http://www.virusbtn.com/resources/glossary/ransomware.xml

[52] http://www.virusbtn.com/resources/glossary/regedit.xml

[53] Microsoft. *Glossary of DIA SDK Terms*. Microsoft, 2012. http://msdn.microsoft.com/en-us/library/5e6y0hkw%28v=vs.80%29.aspx

[54] http://en.wikipedia.org/wiki/Position-independent_code

[55] http://www.virusbtn.com/resources/glossary/rogue_antimalware.xml

[56] Gennari, Jeff. "Building a Malware Analysis Capability." *CERT Podcast Series*. Software Engineering Institute, Carnegie Mellon University, 2011. http://www.cert.org/podcast/show/20110712gennari.html

[57] SAND Network Systems. *SandNet*. SAND Network Systems, 2012. http://www.sandsys.com/products/sandnet/index.htm

[58] Osterman, Larry. "What Is This Thing Called, SID?" *Larry Osterman's WebLog*, Sep. 1, 2004. Microsoft, 2012. http://blogs.msdn.com/b/larryosterman/archive/2004/09/01/224051.aspx

[59] http://www.virusbtn.com/resources/glossary/shellcode.xml

[60] Kornblum, Jesse. *ssdeep – Latest Version 2.9*. http://ssdeep.sourceforge.net/ (2012).

[61] http://www.virusbtn.com/resources/glossary/typosquatting.xml

[62] http://www.virusbtn.com/resources/glossary/vgrep.xml

[63] Kissel, Richard, ed. *Glossary of Key Information Security Terms* (NIST IR 7298, Revision 1). National Institute of Standards and Technology (NIST), 2011. http://csrc.nist.gov/publications/nistir/ir7298-rev1/nistir-7298-revision1.pdf

[64] http://www.virusbtn.com/resources/glossary/wabbit.xml

[65] http://en.wikipedia.org/wiki/W%5EX

[66] Alvarez, Victor Manuel. "YARA in a Nutshell," *yara-project: A Malware Identification and Classification Tool*. Google, 2012. http://code.google.com/p/yara-project/

[67] http://www.virusbtn.com/resources/glossary/zip_bomb.xml

References

URLs are valid as of the publication date of this document.

[Alvarez 2012]
Alvarez, Victor Manuel. "YARA in a Nutshell," *yara-project: A Malware Identification and Classification Tool*. Google, 2012. http://code.google.com/p/yara-project/

[Anley 2007]
Anley, Chris; Heasman, John; Lindner, Felix; & Richarte, Gerardo. *The Shellcoder's Handbook: Discovering and Exploiting Security Holes*. Wiley, 2007.

[ARM Technology 2009]
ARM Security Technology. *Building a Secure System Using TrustZone® Technology*, Glossary-2. ARM Limited, 2009. http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf

[Atkins 2008]
Atkins, Sue & Rundell, Michael. *The Oxford Guide to Practical Lexicography*. Oxford University Press, 2008.

[Borda 2011]
Borda, Monica. *Fundamentals in Information Theory and Coding*. Springer-Verlag, 2011.

[Brown 1996]
Brown, Lawrie. *Mobile Code Security*. School of Computer Science, Australian Defence Force Academy, 1996.

[Caralli 2011]
Caralli, Richard A.; Allen, Julia H.; & White, David W. *The CERT® Resilience Management Model (CERT®-RMM): A Maturity Model for Managing Operational Resilience*. Addison-Wesley, 2011.

[Chikofsky 1990]
Chikofsky, Elliot J. & Cross, James H., II. "Reverse Engineering and Design Recovery: A Taxonomy." *IEEE Software 7*, 1 (January 1990): 13-17.

[Distler 2007]
Distler, Dennis. *Malware Analysis: An Introduction*. SANS Institute, 2007.

[Eilam 2005]
Eilam, Eldad. *Reversing: Secrets of Reverse Engineering*. Wiley, 2005.

[Faith 1997]
Faith, R. & Martin, B. *A Dictionary Server Protocol*. Internet Engineering Task Force RFC 2229, 1997.

[Fletcher 2012]
Penney, Fletcher. *MultiMarkdown*. http://fletcherpenney.net/ (2012).

[Gennari 2011]
Gennari, Jeff. "Building a Malware Analysis Capability." *CERT Podcast Series*. Software Engineering Institute, Carnegie Mellon University, 2011. http://www.cert.org/podcast/show/20110712gennari.html

[Ishaq 2010]
Ishaq, A.F.M. "Anti-Reversing as a Tool to Protect Intellectual Property," 248-252. *Second International Conference on Engineering Systems Management and Its Applications*. Sharjah, United Arab Emirates, Mar. 2010. IEEE 2010.

[Kissel 2011]
Kissel, Richard, ed. *Glossary of Key Information Security Terms* (NIST IR 7298, Revision 1), 205. National Institute of Standards and Technology (NIST), 2011. http://csrc.nist.gov/publications/nistir/ir7298-rev1/nistir-7298-revision1.pdf

[Kornblum 2012]
Kornblum, Jesse. *ssdeep – Latest Version 2.9*. http://ssdeep.sourceforge.net/ (2012).

[Lawson 2007]
Lawson, Nate. "Mesh Design Pattern: Hash-and-Decrypt." *root labs rdist,* April 9, 2007. http://rdist.root.org/2007/04/09/mesh-design-pattern-hash-and-decrypt/

[Ligh 2010]
Ligh, Michael; Adair, Steven; Hartstein, Blake; & Richard, Matthew. *The Malware Analyst's Cookbook*. Wiley, 2010.

[Lyashko 2012]
Lyashko, Alexey. *System Programming Blog*. http://syprog.blogspot.com (2012).

[Microsoft 2012a]
Microsoft. *Glossary of DIA SDK Terms*. Microsoft, 2012. http://msdn.microsoft.com/en-us/library/5e6y0hkw%28v=vs.80%29.aspx

[Microsoft 2012b]
Microsoft. "Object Names." *Windows Dev Center – Hardware*. Microsoft, 2012. http://msdn.microsoft.com/en-us/library/windows/hardware/ff557762%28v=vs.85%29.aspx

[MITRE 2010]
MITRE. *Science of Cyber-Security* (JSR-10-102). MITRE Corporation, 2010.

[Mundie 2012]
Mundie, David & Ruefle, Robin. "Building an Incident Management Body of Knowledge." *Seventh International Conference on Availability, Reliability, and Security (ARES 2012)*, Prague, Aug. 20-24, 2012. IEEE Computer Society, 2012.

[Murray 1977]
Murray, K.M. Elisabeth. *Caught in the Web of Words: James A.H. Murray and the Oxford English Dictionary*. Yale University Press, 1977.

[NIST 2012]
National Institute of Standards and Technology (NIST). *National Software Reference Library*. NIST, 2012. http://www.nsrl.nist.gov/

[Osterman 2004]
Osterman, Larry. "What Is This Thing Called, SID?" *Larry Osterman's WebLog*, Sep. 1, 2004. Microsoft, 2012. http://blogs.msdn.com/b/larryosterman/archive/2004/09/01/224051.aspx

[OWASP 2009]
Open Web Application Security Project (OWASP). *Code Injection*. http://www.owasp.com/index.php/Code_Injection (2009).

[Parker 2010]
Parker, Timothy. *Protecting Cryptographic Keys and Functions from Malware Attacks*. University of Texas at San Antonio, 2010.

[Preda 2007]
Preda, Mila Dalla. *Code Obfuscation and Malware Detection by Abstract Interpretation*. Università degli Studi di Verona, 2007.

[Riecks 2012]
Riecks, David. *Controlled Vocabulary*. http://www.controlledvocabulary.com (2012).

[SAND 2012]
SAND Network Systems. *SandNet*. SAND Network Systems, 2012. http://www.sandsys.com/products/sandnet/index.htm

[SEI 2012]
Software Engineering Institute. *Malware Analysis Apprenticeship*. Software Engineering Institute, Carnegie Mellon University, 2012. http://www.sei.cmu.edu/training/P88.cfm

[Symantec 2012]
Symantec. *Trojan.Dropper*. http://www.symantec.com/security_response/writeup.jsp?docid=2002-082718-3007-99 (2012).

[Virus Bulletin 2012]
Virus Bulletin. *Glossary*. http://www.virusbtn.com/resources/glossary/index (2012).

[Weisstein 2012]
Weisstein, Eric W. *Zipf's Law*. http://mathworld.wolfram.com/ZipfsLaw.html (2012).

[Wikipedia 2012]
Wikipedia. http://www.wikipedia.org/ (2012).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	February 2013	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
The MAL: A Malware Analysis Lexicon	FA8721-05-C-0003

6. AUTHOR(S)

David A. Mundie, David M. McIntire

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2013-TN-010

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFLCMC/PZE/Hanscom Enterprise Acquisition Division 20 Schilling Circle Building 1305 Hanscom AFB, MA 01731-2116	n/a

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

The lack of a controlled vocabulary for malware analysis is a symptom of the field's immaturity and an impediment to its growth. Malware analysis is a splintered discipline, with many small teams that for cultural reasons do not, or cannot, readily communicate among themselves; this condition encourages the growth of many local dialects. This report presents the results of the Malware Analysis Lexicon (MAL) initiative, a small project to develop the discipline's first common vocabulary.

14. SUBJECT TERMS	15. NUMBER OF PAGES
malware, malware analysis, vulnerability analysis, lexicography, dictionary, vocabulary	47

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL