Advanced Project in Computer Science: System-Calls Based Dynamic Analysis Intrusion Detection System with Configurable Machine-Learning Classifier

Ishai Rosenberg

The Open University of Israel, Raanana, Israel

Abstract. In this project we implement an IDS based on a configurable machine learning classifier, using system calls executed by the inspected code during its run-time in a sandbox as features. We describe the rationale and design decisions of the IDS implementation, and measure the effectiveness of the IDS when detecting malware it didn't encounter before. We then use this IDS to compare between different machine learning classifiers to find the fittest.

Table of Contents

1 Introduction

The constant rise in the complexity and number of software security threats makes intrusion detection systems (IDS) developers invest large amounts of resources in-order to improve their detection rates.

In this report, we refer to IDS as a tool to detect and classify malware.

Past IDS generally used two methods of malware detection:

1. Signature-based Detection - Searching for known patterns of data within the executable code. A malware, however, can modify itself to prevent a signature match, for example: by using encryption. Thus, this method can be used to identify only known malware.
2. Heuristic-based Detection - Generic signatures, including wild-cards, which can identify a family of malware. Thus, this method can identify only variants of known malware.

Machine-learning can be used in-order to extend the IDS capabilities to classify software unseen before as malicious or benign by using static or dynamic features of the code.

Using system calls sequences as a classifier's input was reported in [1], [3].

Our project focuses on creating an IDS based on a binary (i.e. benign or malware) classifier, of a user-configured type, where the classifiers features are the system calls executed by the inspected code.

Our project has several contributions:

- While there are open source tools that implement similar (although not identical) functionality for *NIX platforms, we would implement this functionality on the Windows platform, which better fit the profile of a computer user today - and which is the target of a significant portion of today malware.
- All the code, except for the sandbox, would be open-source code. This would allow easy modification of the system, in-order to custom-fit it for future research (e.g., by changing the machine-learning algorithm, etc.)
- While the system-call recorder would be Windows-specific, all other parts of the code would be platform-independent. This would ease the modification of the IDS in-order to use it for research under different platforms, e.g., in a Unix platform, by using *strace()* instead-of our custom system call recorder.
- This system has a possibility to choose the machine learning algorithm to be used by it (e.g. decision tree, SVM, etc.), facilitating the comparison of different algorithms. This option is not available in any of the possible open source or binary tools available in the web that I know of.

2 Background and Related Work

2.1 Machine Learning Binary Classifier

The usage of system calls to detect abnormal software behavior has been introduced in [3]. The authors scanned traces of normal behavior and build up

a database of characteristic normal patterns, i.e. observed sequences of system calls. They defined normal behavior in terms of short n-grams of system calls. A small fixed size window and "slide" it over each trace, recording which calls precede the current call within the sliding window. Then they scanned new traces that might contain abnormal behavior, looking for patterns not present in the normal database: System call pairs from test traces are compared against those in the normal profile. Any system call pair (the current call and a preceding call within the current window) not present in the normal profile is called a mismatch. A system call is defined as anomalous if there are any mismatches within its window. If the number of anomalous system calls within a time frame exceeds a certain threshold - an intrusion is reported. The authors also introduced open source tools to report such anomalies ([1] and [2]).

For example, consider a mail client that is under attack by a script that exploits a buffer overrun, adds a backdoor to the password file, and spawns a new shell listening on port 80. In this case, the system call trace will probably contain a segment looking something like: *open(), write(), close(), socket(), bind(), listen(), accept(), read(), fork()*. Since it seems unlikely that the mail client would normally open a file, bind to a network socket, and fork a child in immediate succession, the above sequence would likely contain several anomalous sub-traces, and thus this attack would be easily detected.

Data mining techniques and machine learning algorithms such as Naive Bayes have also been used with the Windows platform ([31]). Other machine learning algorithms, such as decision trees, SVM, boosted trees, Bayesian Networks and Artificial Neural Networks were used and compared to find the best classification algorithm - with inconclusive results (e.g.: [30] and [29] chose boosted decision trees as the most accurate method, [28] chose decision trees, etc.) which are affected by the exact samples in the training set and their number, the type of the feature set used, etc.

In-order to classify code as benign or a malware, machine-learning algorithms use distinct features as input. Types of features that have been used to classify software are either extracted statically (i.e. without running the inspected code): byte-sequence (n-gram) in the inspected code (as in [30] and [29]), APIs in the Import Address Table (IAT) of executable PE headers (as in [6]), or disassembly of APIs in the executable (as in [21]). The features can also be extracted dynamically (i.e. by running the inspected code): CPU overhead, time to execute (e.g., if a system has an installed malware driver being called whenever accessing a file or a directory to hide the malware files, then the additional code being run would cause file access to be longer than usual), memory and disk consumption (as in [4] and [5]), machine-language op-codes sequence executed (as in [20]), executed system calls sequence (as in [1] and [2]) - or even non-consecutive executed system calls sequence (as in [7]).

While gathering the inspected code's features using static analysis has the advantage of not needing to run a possible malicious code (as done, for example, in [6] and [21]) - it has a main disadvantage: since the code isn't being run - it might not reveal its "true features" (as shown in [19]). For example, if you

inspect the APIs in the Import Address Table (IAT) of executable PE headers (as done in [6]), you would miss APIs that are being called dynamically (using *LoadLibrary()\GetProcAddress()* in Windows and *dl_open()\dl_sym()* on Unix). If you look for byte-sequence (or signatures) in the inspected code (as done in [30]) - you might not be able to catch polymorphic malware, in which those signatures are either encrypted or packed and decrypted only during run-time, by a specific bootstrap code. Similar bootstrap code can be used for "legitimate", benign applications either, so detection of such code is not enough. Other limitations of static analysis and techniques to counter it appear in [19].

Thus, we decided to use dynamic analysis, which reveals the true features of the code. Obviously, a malware can still try to hide, e.g., by detecting if some other application (the IDS) is debugging it or otherwise monitoring its features and not operate its malicious behavior at such cases, as done, for example, by the "Blue Chicken" technique presented in [23]. However, it is much harder for a malware writer to do (correctly) and in the end - in-order to operate its malicious functionality - a malware must reveal its dynamic features, during run-time. However, those features can be altered in a way that would fool an IDS, as have been shown in my thesis.

2.2 Sandboxing Dynamic Analysis IDSs

The main issue with using a dynamic analysis IDS is the fact that it must run the inspected code, which might harm the hosting computer. In-order to prevent a damage to the host during the inspection of the code, it's common to run the code in a sandbox: a controlled environment, which isolates between the (possibly) malicious code to the rest of the system, preventing damage to the latter. Any harmful modifications done in the sandbox can usually be monitored and reverted, if needed. In-order to avoid the malicious code from detecting that it's running on a sandbox (and thus is being monitored), the sandbox should be as similar as possible to the actual system. The isolation can be done either at the application-level, meaning that the malicious code is running on the same operating system as the rest of the system, but its system calls effect only a quarantined area of the system, e.g., registry key modifications done by the code are being redirected to different keys (as done in [24]), on the operating-system level, meaning the operating system is isolated (and thus damage to that operating system does not effect the host operating system) - but the processor is the same (as done in [9]), or at the processor level, meaning all machine instruction are emulated by a software called an emulator (like QEMU, [12]).

CWSandbox ([9]) is an operating-system level sand-boxing tool. During the initialization of an inspected binary executable, CWSandbox's dll is injected into its memory to carry out API hooking. This dll intercepts all API calls and reports them to CWSandbox. The same procedure is repeated for any child or infected process. CWSandbox and the malicious code are being executed inside a virtual machine (or VM, a software implementing a completely isolated guest operating system installation within a normal host operating system) based on VMware Server and Windows XP as guest system. After each code analysis, the

VM is reverted to a clean snapshot. An XML report of all executed system calls is being generated by this tool.

TTAnalyze ([8]) uses processor level sand-boxing to run the unknown binary together with a complete operating system in software. Thus, the malware is never executed directly on the processor. While both tools generate a system calls report, some of the major differences between TTAnalyze and CWSandbox are:

- TTAnalyze uses the open-source PC emulator QEMU rather than a virtual machine, which makes it harder for the malware to detect that it's running in a controlled environment (since all machine instruction are emulated by the software it should be transparent to the analyzed code running inside the guest OS) .
- TTAnalyze does not modify the program that it executes (e.g., through API call hooking), making it more difficult to detect by malicious code via code integrity checks. This is done by using adding a callback after each basic block (a sequence of one or more instructions that ends with a jump instruction or an instruction modifying the static CPU state in a way that cannot be deduced at translation time) translated by QEMU.
- TTAnalyze monitors calls to native kernel functions (undocumented internal implementation, susceptible to changes) as well as calls to Windows API functions (documented functions that call internally to the native functions). Malware authors sometimes use the native API directly to avoid DLL dependencies or to confuse virus scanner's operating system simulations.
- TAnalyze can perform function call injection. Function call injection allows TTanalyze to alter the execution of the program under analysis and run TTAnalyze code in its context. This ability is required in certain cases to make the analysis more precise (for example, the CreateFile API could both create a new file or open an existing one, so a code that checks whether the file existed before the call should be injected before such a call in the analyzed code in-order to properly log to API call effect).

Other dynamic analysis tools are surveyed at [33].

While an emulator-based sand-boxing technique might be harder to detect - it can be done, as shown in [13], [14] and [15]. Furthermore, the significant performance degradation (up to 20 times slower, as described in [16]) makes such system vulnerable to timing attacks (as described, for example, in [17] and [18]).

3 Problem Description

The general problem can be defined formally as follows:

Given the traced sequence of system calls as the array *sys_call*, where the cell: *sys_call[i]* is the i-th system call being executed by the inspected code (*sys_call[1]* is the first system call executed by the code),

Define the IDS classifier as: *classify(benign_training_set, malicious_training_set, inspected_code_sys_calls)*, where *inspected_code_sys_calls* is the inspected code's system calls array, *benign_training_set* is a set of system calls arrays used to train the classifier with a known benign classification and *malicious_training_set* is a set of system calls arrays used to train the classifier with a known malicious classification. *classify()* returns the classification of the inspected code: either benign or malicious.

3.1 Evaluating the Classification Algorithm

The quality of our IDS would be determined by two factors (P is the probability\frequency of occurrence):

1. We would like to minimize the false negative (FN) rate of the IDS, i.e. to minimize P(*classify(benign_training_set, malicious_training_set, malicious_inspected_code_sys_calls)* = benign).
2. We would like to minimize the false positive (FP) rate of the IDS, i.e. to minimize P(*classify(benign_training_set, malicious_training_set, benign_inspected_code_sys_calls)* = malicious).

In-order to take into account true and false positives and negatives, we would try to maximize the Matthews correlation coefficient (MCC), which is used in machine learning as a measure of the quality of binary classifications:
$MCC = \frac{TP*TN-FP*FN}{\sqrt{(TP+FP)*(TP+FN)*(TN+FP)*(TN+FN)}}$[1] ([41]). In the following sections we show the main parts of our IDS: the IDS usage flow (section 4), the implementation of the monitoring system and the classifier including theoretical background about the different classifiers implemented (section 5) and the comparison of the different classifiers by the criteria given at this section (section 6).

4 The IDS Usage Flow

The flow of our IDS is as follows (numeric steps are user-initiated and letters are code-initiated):

1. The user calls creates an IDS instance in a Python shell, giving it the classifier type as a parameter:
   my_ids=IDS("Linear SVM")
   (a) The IDS loads a file containing the serialized classifier trained using the training set. The file name is hard-coded and is determined by the classifier's type, although user defined file can also be given, if needed.

[1] TP - True Positive - A malware that has been classified as malicious. FP - False Positive - A malware that has been classified as benign. TN - True Negative - A benign software that has been classified as benign. FN - False Negative - A benign software that has been classified as malicious.

2. The user runs the inspect command, giving it a file path to analyze: *my_ids.inspect(r"c:\temp\file_to_inspect.exe")*
   (a) The IDS opens a sandbox, if one is not already up, from previous user commands. The path to the sandbox to open is taken from the IDS configuration file.
   (b) The IDS reverts the sandbox to a clean state, to start the inspection with "a clean slate". The name of the snapshot to revert to is taken from the IDS configuration file.
   (c) The IDS logs-in into the sandbox with the user name and password, which are taken from the IDS configuration file.
   (d) The IDS copies the required files (the system calls recorder and the file to inspect) into the sandbox. The required file paths are taken from the IDS configuration file.
   (e) The IDS runs the system calls recorder inside a new shell (which path is taken from the IDS configuration file) inside the sandbox for a fixed amount of time.
   (f) The system calls recorder runs the inspected file (given as one of its arguments by the IDS) and prints the system calls specified in its own configuration file (also copied to the sandbox in the last step, since it appears in the IDS configuration file) when they're being executed. This output (i.e., all recorded system calls) is being redirected to the output file path inside the sandbox (the path is taken from the IDS configuration file). Note that infection or damage can happened only within the sandbox itself, since the sandbox is disconnected from the internet.
   (g) After the fixed amount of inspected code running time has passed, the IDS terminates the system calls recorder's and the inspected file's processes, if it created a new one (if they didn't exit before).
   (h) The IDS copies the system calls recorder output file from the sandbox to the host computer. The path is taken from the IDS configuration file.
   (i) The IDS parses the system calls recorder output file to a XML file, to maintain a unified input format, regardless of the system call recorder output file format.
   (j) The IDS parses the XML file, which was generated in the previous step, and extract the system calls being called by the inspected code, in a label feature format, e.g.: *sys_call[1] = NtCreateFile*.
   (k) The IDS transforms all label features to a numeric format of vector of bits, each one represents if a specific feature (e.g.: *sys_call[2] = NtClose*) exists or not in the executed code. Each feature from the last step would be transformed to a single 1-bit in the vector of all possible features, in-order to fit the classification algorithms.
   (l) The IDS performs feature selection on the features extracted, based on the most significant features, as calculated for the training set. See section 5.4.
   (m) The selected features are the input of the classifier.
   (n) The classifier returns a classification of the inspected file based on the selected features: benign or malicious.

5 IDS Implementation

From the above flow, we see that our IDS has 5 main parts:

1. A sand-boxing mechanism that would run the inspected code inside a virtual machine without an internet connection (to prevent the possibility of infecting other machines) for a limited amount of time.
2. Feature extraction - Recording all the system calls made by the inspected code while it's running.
3. Feature parsing - Converting the system calls recorder output to a data set that fits our classifier.
4. Feature selection and classification - Feeding the system calls sequence to a machine-learning binary classifier and receiving a classification for the inspected code: malicious or benign.
5. A wrapper which provides the user interface to the IDS.

Those parts are completely interdependent, and each one can be replaced without affecting the others, as-long-as their interfaces remains the same.

The class diagram of the entire IDS is shown in Figure 1.

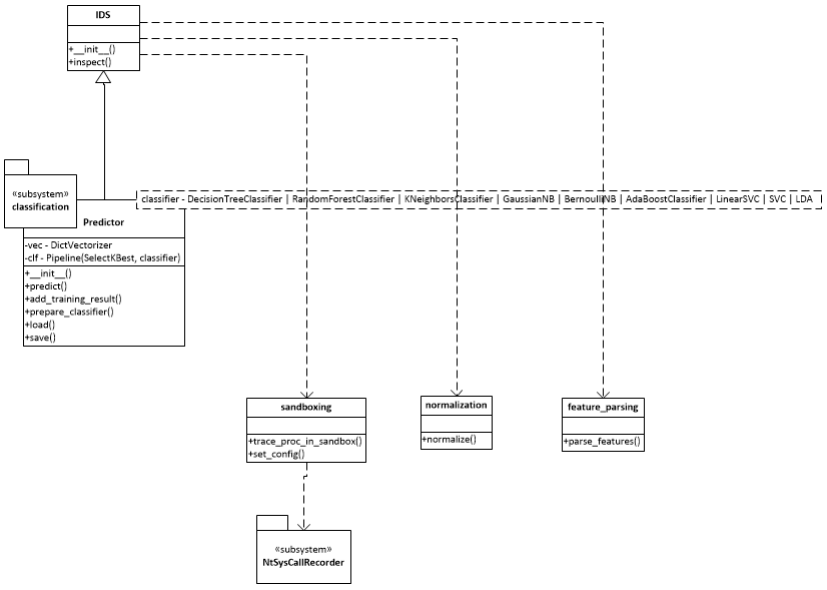


Fig. 1. The IDS Python Class Diagram

Note that only the Python modules external interfaces are specified here. The *NtSysCallRecorder* detailed class diagram is shown in Figure 2.

The IDS doesn't have a UI - only a command-line interface. This is due-to the fact that it was designed as a research tool with an easy API to query various

properties of the IDS (e.g., the detection rate) - and not as a tool for end-users. While a UI and a on-line scanning mode could be added, a system based on dynamic analysis doesn't fit a day-to-day continuous usage: It's unreasonable that a user would have to wait 10-15 seconds for each file that is being analyzed - especially when he copies 100 files or more.

5.1 Sand-Boxing Mechanism

As mentioned in the end of section 2, in-order to implement a dynamic analysis IDS that would sandbox the inspected code effects, we have chosen to use VMWare Workstation[2], a commonly used virtual machine software, where changes made by a malicious code can be reverted. In-order to automate the operations inside the sandbox (e.g., copying files to or from the VM, reverting to a snapshot, etc.), we use *vixpy*[3], which is a wrapper around the VIX API[4].

Our IDS executes the inspected code on a virtual machine with Windows XP SP3 OS without an internet connection (to prevent the possibility of infecting other machines).

Windows OS better suits our needs than Unix variants, used by available open-source tools (e.g., [1] and [2]), since most malware target it[5].

The inspected executables were run for a period of 10 seconds - and then forcefully terminated, if the process didn't exit before that by itself.[6] This amount of time is enough to record significant amount of system calls (about 10,000 recorded system calls per executable on average, and a maximum amount of more than 50,000 recorded system calls) - but not too much for a system with

[2] Initially, I wanted to use VirtualBox (https://www.virtualbox.org/) as the sandbox. The reason is that this is an open-source tool, which can be freely downloaded and its code can be tweaked, if desired. However, VirtualBox does not have an equivalent to the VIX API of VMWare which suits our needs (e.g., allows log-in to the guest OS, loading a snap-shot, etc.). I tried to use BOINC (https://boinc.berkeley.edu/trac/wiki/VirtualBox) - but its suport on VirtualBox is far from perfect, and after trying to stabilize it for a long time - I decided that the effort needed was too great, and since this was not a main feature of the project - I decided to switch to VMWare, which VIX API is far more stable and mature.

Also note that, while the free VMWare Player does support the VIX API - it supports only a limited subset, which is not enough for our needs (e.g., it can't revert to a snap-shot, etc.)

Thus, only VMWare Workstation is supported in the IDS code.

[3] https://github.com/dshikashio/vixpy

[4] https://www.vmware.com/support/developer/vix-api/

[5] https://www.daniweb.com/hardware-and-software/microsoft-windows/viruses-spyware-and-other-nasties/news/310770/99-4-percent-of-malware-is-aimed-at-windows-users

[6] Tracing only the first seconds of a program execution might not detect certain malware types, like "logic bombs" that commence their malicious behavior only after the program has been running some time. However, this can be mitigated both by classifying the suspension mechanism as malicious or by tracing the code operation throughout the program execution life-time, not just at the beginning.

8-16GB of RAM to calculate the classifier, with this amount of features in a reasonable amount of time.

Implementation The python module in-charge of the sand-boxing mechanism is *sandboxing*. It has two interface method:

- *trace_proc_in_sandbox()* - Performs steps 2.(a)-2.(h) in section 4.
- *set_config()* - Sets the IDS configuration file (used by *trace_proc_in_sandbox()*) path. The configuration file parameters are specified in Appendix D. In-order to allow modifications in the configuration file format without affecting the *sandboxing* module code, we implemented an abstraction layer called *config*, which you can use to access the configuration file fields (e.g.: *config.tracer_settings.tracer_executable*) without being aware of the configuration file format specified in Appendix D.

Note that the *sandboxing* module can be modified, e.g., to use an emulator, such-as QEMU ([12]), instead-of VMWare Workstation, without any effect on the rest of the system, as long as the external interface (which, unlike the internal methods, doesn't contain any VM-specific parameters) is intact.

5.2 Feature Extraction: System Calls Recorder

While we are focusing on the Windows OS, our IDS design is cross-platform, written in Python[7], a cross-platform programming language, and designed for modularity. The only part which is platform-specific is this one: the system calls recorder. For example, our Windows system-calls recorder could be replaced by *strace()*[8] in-order to operate on Linux OS.

The system calls recorder we have used for Windows records the Nt* system-calls[9] in ntdll.dll. The usage of this low layer of system calls was done in-order to prevent malware from bypassing Win32API (e.g. *CreateFile()*) recording by calling those lower-level, Nt* APIs (e.g. *NtCreateFile()*). We have recorded 444 different system calls, such-as *NtClose(), NtWaitForMultipleObjects(),* etc. The full recorded system calls list can be found on Appendix C.

Implementation Several ways to implement API hooking, needed by the system calls recorder, were considered:

- IAT (import address table) patching - We can modify the import table of the process we want to monitor, so the API would be recorded before calling the actual API, by providing an address of a new function that would log the original function - and then call it. The problem in this technique is that it only monitors statistically-linked functions, meaning, not functions that were called dynamically, by a *LoadLibrary()\GetProcAddress()* combination.

[7] https://www.python.org/
[8] http://en.wikipedia.org/wiki/Strace
[9] http://undocumented.ntinternals.net/

We can reduce the number of missed APIs by patching the import table of each DLL loaded into the monitored process. We can also return our own function pointer from a call to *GetProcAddress()* and thus ease the monitoring of dynamically-called API. In conclusion, this method doesn't cover all available API calls and fixing it is complex and not scalable.

- EAT (export address table) patching - By changing the export table of the DLL in-which our monitored API exists, each such call can be logged before the actual API is being called, again, by replacing the original API with a function that logs the call before actually making it. There are two major issues with this technique: 1) It monitors every process that calls the hooked API - and not just the inspected process. Logging all of those API calls could be a huge performance bottleneck on the system. 2) Saving the monitored process path in a global location, such that would be accessible by all loaded Dll (so only the specific process would be monitored and not the entire system) is dangerous, since a malware might change it (e.g., to a benign process), harming the inspection process. The same goes for the log itself.
- Detours - As mentioned in [10], this method involves modifying the code of the API in the monitored process memory at run-time, inserting a JMP instruction at the beginning of the API code to the logging code, performs the logging - and then JMP back to the original API's code. A problem with this method is that since there is a modification of the inspected code memory, it can perform an integrity check (e.g., comparing the hash value of the memory address space to the file system) and thus - it's easy for it to detect that it's being monitored (and cease its malicious operation).
- Proxy\Trojan DLL - Replacing the DLL we want to monitor (here: NtDll.dll) with a different dll with the same entry-points (same arguments, etc.), which record the system call before calling the original API. Again, this is a system-wide mechanism, so you have the same issues as with EAT patching.
- Kernel hooks - Modifying the SSDT (a global table that contains the Win32 SubSystem functions' addresses) or the IDT (a global table that contains the addresses of the interrupts handling procedures, in-order to handle *INT 2e* or *SYSENTER* calls) was specified, e.g., in [43]. The problem here is that a kernel hook is extremely OS-specific - and might be needed to be modified even between service packs of the same Windows release. In addition, as before, this is a system-wide hook, which is not process-specific.
- Debugging events - Running the inspected code via a custom-made debugger which would set breakpoints and monitor the Nt* API calls from the inspected code in those breakpoints.

We have decided to use the debugger option due-to the following reasons:

- It is available "by-design" and it is a documented API. Thus, we don't use here a method that might be broken or modified in future service packs or OS versions.
- It is process-specific hook, as needed, and not system-wide, meaning we have only the smallest performance degradation possible - and we don't get any

irrelevant system calls, which we then need to filter using additional, error-prone logic.

– A debugger has read and write privileges to the debugged process - so it's harder to "fool" it.

– The implementation is easier than other methods (again, this is a documented API).

While, as mentioned in section 2.1, a malware can try to hide, e.g., by detecting if some other application (the IDS) is debugging it, the fact that the process is being debugged (e.g., by the API *IsDebuggerAttahed()*) can be concealed by modifying the PEB, as written, for example, in http://undocumented.ntinternals.net/source/usermode/undocumented functions/nt objects/process/peb.html.

Unlike the rest of the system, which is written in cross-platform Python, the system call recorder is platform-specific, and call platform specific APIs in C. Thus, it is written in C++, in-order to easily access those APIs, but still be written in an object-oriented manner. It is implemented as a console application, which gets the file path to trace as its only command-line argument. It can be used independently from the other parts of the system. The only reference to the system calls recorder's path is in the IDS configuration file (to keep a loose coupling from the rest of the system), from where it is being read, by the *sandboxing* module, which operates it on the inspected code file inside the sandbox.

The class diagram of the system calls recorder, *NtSysCallRecorder*, is shown in Figure 2.

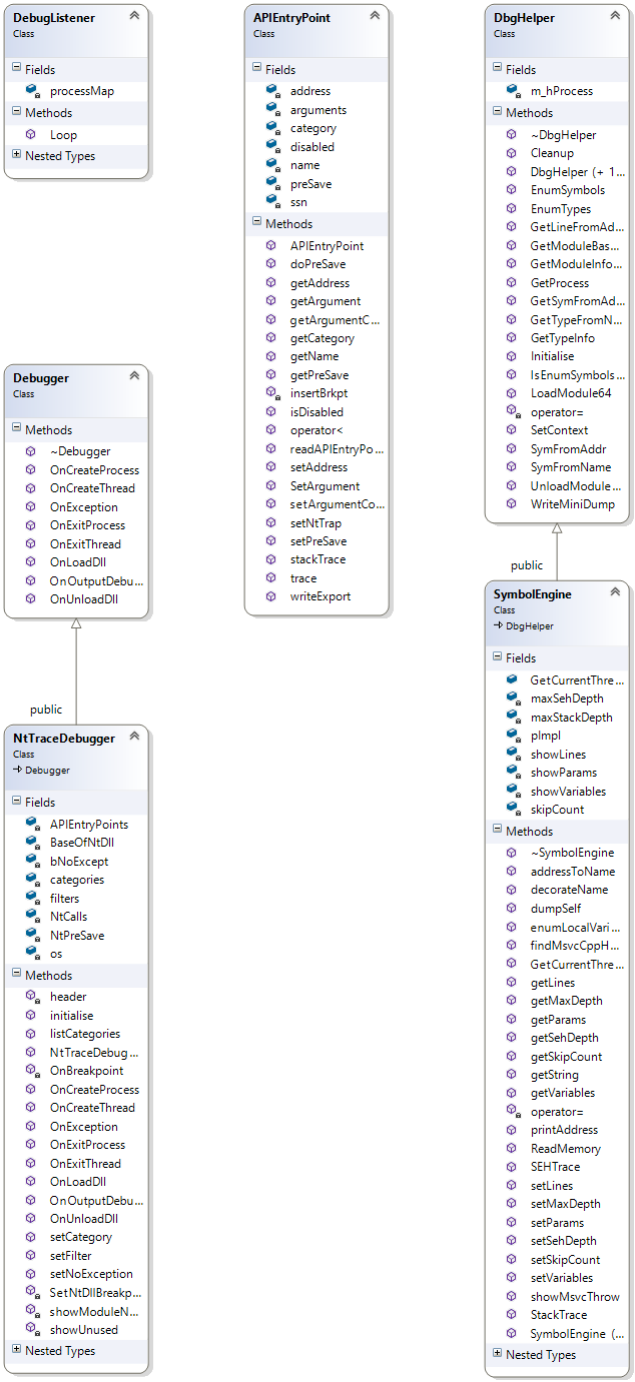


Fig. 2. The System Calls Recorder Class Diagram

The main classes in the system call recorder are:

- *DebugListener* - This is a listener to the debug events (e.g., new thread, DLL load, etc.) generated by the inspected code when it is being run through the system calls recorder.
- *Debugger* - This is the interface for the callbacks being called by the *DebugListener* when debug events happens.
- *NtTraceDebugger* - This is the implementation of the *Debugger* interface for Nt API entry points in NtDll.dll.
- *APIEntryPoint* - This class represents an Nt API entry point (loaded from the system call recorder configuration file), where the system call recorder can set breakpoints to debug the API call, as needed.
- *DbgHelper* - This is a wrapper around the DbgHelp API[10], which allows us, e.g., to get the function name from an address in the debugged process address space, using the symbols for Windows OS.
- *SymbolEngine* - This class implements higher-level methods, such-as current local variables, function parameters and full stack-trace information, using the lower-level *DbgHelper* interface.

The *main()* function simply reads the APIs to monitor from the system calls recorder configuration file, create an *APIEntryPoint* instance for each one, attach a *DebugListener* instance as the debugger of the inspected code, pass the debug events to a *NtTraceDebugger* instance that logs the proper information, using *SymbolEngine* to extract the relevant information needed to be recorded.

An output example of the output generated by the system calls recorder is available in Appendix E.

5.3 Feature Parsing

The parsing part has two main parts::

1. Output normalization - converting the system-calls recorder-specific serialized (i.e. file) output to a serialized generic output. This is done by an abstraction layer that reads the output file generated by redirecting the system calls recorder output to a file, parse each recorded system call name, arguments and return value - and write the information in an output XML file. If we change the system call recorder in the last section - only this abstraction layer should be modified to fit the new output format. An output example of the normalized output file can be found in Appendix F.
2. Feature parsing - reading the normalized output file, generated in the last step - and de-serializing it to a usable format. In-our case this is a simple Python dictionary where the key is the system call position and the value is the system call type (as mentioned in section 5.4, the arguments and return values are not being used by the IDS, although they are available). As an example, if the third system call was *NtCreateFile*, then the feature extracted is: *parsed_features_dict["sys_call3"]="NtCreateFile"*, which is the Python representation for: *sys_call[3] = NtCreateFile.*

[10] https://msdn.microsoft.com/en-us/library/windows/desktop/ms679309

Implementation The Python module in-charge of the output normalization is *normalization*, which has only a single interface method: *normalize()*, which gets the file path to the system call recorder output file, converts it into the normalized XML format - and returns the path to the normalized XML output file, to be used for feature parsing

The Python module in-charge of the feature parsing is *feature_parsing*, which has only a single interface method: *parse_features()*, which takes the normalized output file path, parses and de-serializes it - and returns a Python to be analyzed by the classifier, as mentioned above.

5.4 Feature Selection and Classification: Machine-Learning Binary Classifier

We have implemented the classifier using the Python scripting language and scikit-learn[11]. This library provides a common interface for many data-mining and machine-learning algorithms. Several classifiers were implemented using scikit-learn, as specified in the following sub-sections.

The training set for the binary classifier contains malicious and benign executables. The malicious executables were taken from VX Heaven[12]. They were selected from the Win32 Virus type (and not, e.g., Trojan, Worm or Rootkit). This focus allowed us both to concentrate on a specific mode of action of the malicious code and to reduce the chance of infection of other computers caused by using, e.g., worm samples. Benign executables were taken from both the .\Windows\System32 folder and a collection of benign third-party programs. The number of malicious and benign samples in the set was roughly equal (521 malicious samples and 661 benign samples) to prevent the imbalanced data problem - a bias towards classification with the same value as the majority of the training samples, as presented, for example, in [26].

As features for the decision tree we used the position and the type of the system call, i.e. s*ys_call[i] = system_call_type[k]*, e.g.: *sys_call[3] = NtCreateFile*, which is a different feature than *sys_call[2] = NtCreateFile* and from *sys_call[3] = NtClose*, either. Thus, the number of available feature values was very large (about 850,000)[13]. Therefore, we performed a feature selection ([25]), selecting the 10,000 (best) features with the highest values for the χ^2 (chi-square) statistic ([27]) for the training set, relative to the classifications, and created the decision tree based only on the selected features.

The χ^2 is a statistical hypothesis test in which the sampling distribution of the test statistic is a chi-square distribution when the null hypothesis is true.

[11] http://scikit-learn.org/

[12] http://vxheaven.org/

[13] Note that the fact that we have 444 different system calls type being monitored and 50,000 does not mean each of the $444^{50,000}$ combinations is a feature. The feature is available only if it ever appeared in any part of the training set. E.g.: If *sys_call[1]* was never equal to *NtWaitForMultipleObjects* in the training set then *sys_call[1]=NtWaitForMultipleObjects* would not be a valid feature.

Test statistics that follow a chi-squared distribution arise from an assumption of independent normally distributed data, which is valid in many cases due to the central limit theorem. A chi-squared test can be used to reject the hypothesis that the data are independent. Here, this statistic measures dependence between stochastic variables, so a transformer based on this function "weeds out" the features that are the most likely to be independent of class and therefore irrelevant for classification.

Implementation The Python module in-charge of this step is *classification*, and the main class is *classification.Predictor*.

We start by converting the dictionary we got in the last phase to a data format which our classifiers can work with. Since scikit-learn's classifiers are working with numerical arrays, and not with feature-value dictionaries, we use sklearn.feature_extraction.DictVectorizer class[14] for this conversion, which resides in the *classification.Predictor.vec* data member. It converts all label features to a numeric format of vector of bits, each one represents if a specific feature (e.g.: *sys_call[2] = NtClose*) exists or not in the executed code. Each such feature is transformed to a single 1-bit in the vector of all possible features. While we could have used the system call type as a nominal value (with a possibly different set of possible values for a specific system call position), the scikit-learn algorithms better fit the data format of bit vectors.

This bit-vector is the input for our classifier. It is built as a scikit-learn pipeline[15], which is used to chain multiple estimators into one. This is useful here, where we have a fixed sequence of steps in processing the data: feature selection followed by classification. The first step in the pipeline is the feature selection part, which is implemented by scikit-learn's SelectKBest class[16], where the score function is chi2 (chi-square) and k is 10,000. The classification step is implemented by a scikit-learn's machine-learning binary classifier, chosen by the classifier_type argument given to *classification.Predictor.__init__()* method. This argument is used as the key for the prototype-based factory ([42], page 126) to create the requested classifier - or to deserialize the proper classifier, already trained with the training set and ready to be used, if one already exists. This pipeline resides in the *classification.Predictor.clf* data member.

classification.Predictor has several important external interface method:

- *predict()* - Gets the features dictionary generated in the previous part, mentioned in section 5.3, transforms it to a vector of bits using *Predictor.vec*, performs the feature selection and then the internal scikit-learn classifier's classification using *Predictor.clf* and returns the classification to the user.
- *add_training_result()* - Gets a single features dictionary and its classification, and add them to the classifier's training set.
- *prepare_classifier()* - This method is used to train the classifier with the current training set. It was separated from *add_training_result()* in-order to

[14] http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.DictVectorizer.html
[15] http://scikit-learn.org/stable/modules/pipeline.html
[16] http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html

improve performance: first the user adds all the training set - and only then it trains the classifier on the entire set. If the user called *add_training_result()* followed by *predict()* without calling *prepare_classifier()* in-between - it would be called automatically by *predict()* in-order to make sure that the prediction is done based on the most up-to-date training set.

- *load()* - Gets a file path and load a pre-existing classifier, possibly already trained by a training set. The de-serialization is done using the Python standard library module *cPickle*, the C variant of the *pickle* module, for improved performance. Notice that the *classification.Predictor.__init__()* method calls this method with a file path which is dependent in the classifier's type argument it got, in-order to skip the time needed to re-train the IDS, if the default training set suffices.
- *save()* - Gets a file path and save its classifier, possibly already trained, to this path. The serialization is done using the Python standard library module *cPickle*, the C variant of the *pickle* module, for improved performance.

The *classification.Predictor* abstraction is hiding the internal implementation of the features representation, selection and classification by scikit-learn (e.g., it doesn't inherit from the scikit-learn classifier class), thus allowing the usage of user-generated classifiers or classifiers from a different framework, e.g. Weka[17], without the need to modify the client's code.

The implemented classifiers, including the relevant theoretical background, are presented in the following sub-sections:

5.4.1 Decision Tree Classifier We selected the CART decision tree algorithm[18], similar to C4.5 (J48) decision tree, which was already proven to be a legitimate and even superior algorithm for malware classification ([28]). This tree is the successor of the ID3 decision tree classifier ([22]), which was the first algorithm to use maximum entropy as a decision factor. It is commonly used for implementing malware detection (for example in [20]).

This classifier is easy to explain by an example:

Example 1. An example for a simplified system-calls based decision tree is shown in Figure 3. If the decision tree condition specified in a node is, e.g.: *sys_call[2] =? NtOpenFile*, this means the right child of this node in the decision path is chosen if *sys_call[2]=NtOpenFile,* and the left child is chosen if not.

In this decision tree, if the inspected code trace contains: {*sys_call[1]=NtQueryInformationFile, sys_call[2] = NtOpenFile, sys_call[3]=NtWriteFile, sys_call[4]=NtClose*}, Its path in the IDS's decision tree is: *M=RRL* (=Right-Right-Left), and it would be classified as a malicious.

If the code trace contains: {*sys_call[1]=NtQueryInformationFile, sys_call[2]=NtOpenFile, sys_call[3]=NtAddAtom, sys_call[4]=NtClose*}, the classifier would declare this code as benign, since the decision path would be *B=RRRL*.

[17] http://www.cs.waikato.ac.nz/ml/weka/

[18] http://scikit-learn.org/stable/modules/tree.html

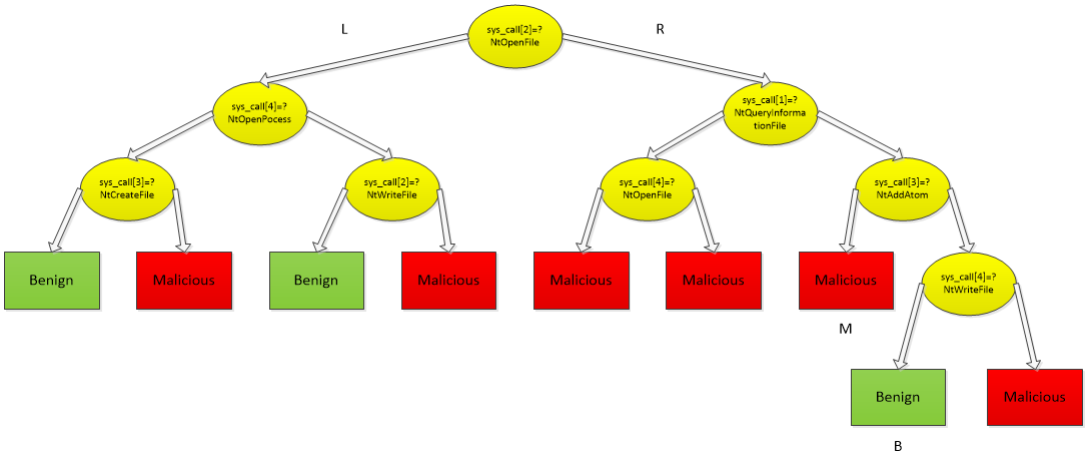


Fig. 3. A System Calls Based Decision Tree

The actual decision tree generated by the training set can be found in appendix A.

A decision tree is invariant under scaling and various other transformations of feature values, is robust to inclusion of irrelevant features, and produces inspectable models. Thus, this is the default classifier of the IDS.

Decision Tree Learning (DTL) Algorithm Our aim is find a small tree consistent with the training examples.

The idea: (recursively) choose "most significant" attribute as root of (sub)tree:

function DECISION-TREE-LEARNING(examples, attributes, parent_examples)

1. *if* examples is empty *then return* PLURALITY-VALUE(parent_examples)
2. *else if* all examples have the same classification *then return* the classification
3. *else if* attributes is empty *then return* PLURALITY-VALUE(examples)
4. *else*
   (a) A$\leftarrow$ $\underset{a \in attributes}{argmax}\ IMPORTANCE(a, examples)$
   (b) tree$\leftarrow$a new decision tree with root test A
   (c) *for each* value v_k of A *do*
      i. exs $\leftarrow$\{e : e $\in$examples *and* e.A $= v_k$\}
      ii. subtree$\leftarrow$DECISION-TREE-LEARNING(exs, attributes $-$A, examples)
      iii. add a branch to tree with label (A $= v_k$) and a subtree subtree
   (d) *return* tree

How do we choose the feature to split by? A good feature splits the examples into subsets that are (ideally) "all positive" or "all negative".

An example of two possible features appear in Figure 4.

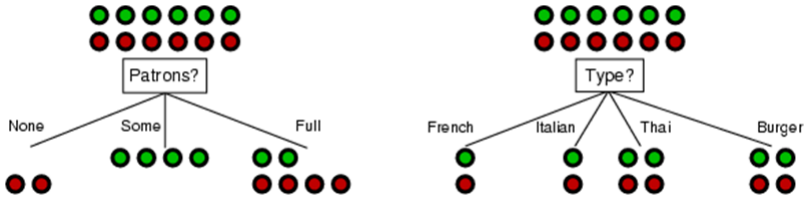


Fig. 4. Example Features to Split-by

In this case, *Patrons?* is a better choice: It minimizes the entropy and increases the uniformity of the classification of the members in the splatted groups, when comparing it to the *Type?* feature.

Entropy To implement Choose-Attribute (IMPORTANCE) in the DTL algorithm, show above, we'd use terms taken from Shannon's information theory:

Information Content (Entropy), ([22]):

H(V)=I(P(v_1), ... , P(v_n)) = $\sum_{i=1}^{n} -P(v_i) \lg P(v_i)$

Where $v_1 \dots v_n$ are the different possible values the random variable V.

The entropy for a training set containing p positive examples and n negative examples:

$I(\frac{p}{p+n}, \frac{n}{p+n}) = -\frac{p}{p+n} \lg(\frac{p}{p+n}) - \frac{n}{p+n} \lg(\frac{n}{p+n})$

Information Gain (IG) A chosen attribute A divides the training set E into subsets E_1, ... , E_v according to their values for A, where A has v distinct values:

$reminder(A) = \sum_{i=1}^{v} \frac{p_i+n_i}{p+n} I(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i})$

Information Gain (IG) or reduction in entropy from the attribute test is:

$IG(A) = I(\frac{p}{p+n}, \frac{n}{p+n}) - reminder(A) = I(\frac{p}{p+n}, \frac{n}{p+n}) - \sum_{i=1}^{v} \frac{p_i+n_i}{p+n} I(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i})$

Thus, we choose the attribute with the largest IG.

Gini Impurity An alternative method to implement Choose-Attribute in the DTL algorithm, shown above, is Gini impurity. This is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it were randomly labeled according to the distribution of labels in the subset. Gini impurity can be computed by summing the probability of each item being chosen times the probability of a mistake in categorizing that item. It reaches its minimum (zero) when all cases in the node fall into a single target category.

To compute Gini impurity for a set of items, suppose $i \in \{1, 2, ..., m\}$, and let f_i be the fraction of items labeled with value i in the set.

$I_G(f) = \sum_{i=1}^{m} f_i(1 - f_i) = \sum_{i=1}^{m} (f_i - {f_i}^2) = \sum_{i=1}^{m} f_i - \sum_{i=1}^{m} {f_i}^2 = 1 - \sum_{i=1}^{m} {f_i}^2$

Implementation The IDS classifier was implemented using scikit-learn's sklearn.tree.DecisionTreeClassifier class[19]. The default metric is: Gini impurity.

[19] http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

5.4.2 Random-Forest Classifier Decision trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets, because they have low bias, but very high variance.

Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set. When splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features. Instead, the split that is picked is the best split among a random subset of the features. As a result of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but, due to averaging, its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better model.

Tree bagging The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners ([35]). Given a training set X = x1, ..., xn with responses Y = y1, ..., yn, bagging repeatedly selects a random sample with replacement of the training set and fits trees to these samples:

1. For b = 1, ..., B:
   (a) Sample, with replacement, n training examples from X, Y; call these Xb, Yb.
   (b) Train a decision or regression tree fb on Xb, Yb.

After training, predictions for unseen samples x' is made by taking the majority vote in the case of decision trees.

As already mentioned, this bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias, or with a minimal increase in total. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic); bootstrap sampling is a way of de-correlating the trees by training them with different training sets.

The number of samples/trees, B, is a free parameter.

Feature Bagging The above procedure describes the original bagging algorithm for trees. Random forests differ in only one way from this general scheme: they use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. This process is sometimes called "feature bagging". The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the decision trees, causing them to become correlated.

Implementation The IDS classifier was implemented using scikit-learn's sklearn.ensemble.RandomForestClassifier class[20]. The default value is B=10 decision trees, using a default metric of Gini impurity.

5.4.3 K-Nearest Neighbors (k-NN) Classifier The input of the k-Nearest Neighbors algorithm consists of the k closest training examples in the feature space ([37]). The output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors. k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification.

A shortcoming of the k-NN algorithm is that it is sensitive to the local structure of the data.

The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples.

In the classification phase, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point.

The used distance metric is Euclidean distance.

Implementation The IDS classifier was implemented using scikit-learn's sklearn.neighbors.KNeighborsClassifier class[21]. The default values are: k=5, distance metric=standard Euclidean metric.

5.4.4 Naïve Bayes (NB) Classifier The classifier is based upon a naïve assumption, by which it is named: Even if features depend upon the existence of others, they independently contribute to the classification ([36]).

A pro of this classifier: It requires a small amount of training data to estimate the parameters (means and variances of the variables) necessary for classification

A con of this classifier: Bayes classification is outperformed by approaches such as RFA and SVM (as mentioned in [11]).

The NB Probability Model naive Bayes is a conditional probability model: given a problem instance to be classified, represented by a vector $\mathbf{x} = (x_1, \ldots, x_n)$ representing some n features (dependent variables), it assigns to this instance probabilities

$p(C_k|x_1, \ldots, x_n)$

for each of k possible outcomes or classes.

Using Bayes' theorem, under the above independence assumptions, the conditional distribution over the class variable C is:

$p(C_k|x_1, \ldots, x_n) = \frac{1}{Z} p(C_k) \prod_{i=1}^{n} p(x_i|C_k)$

[20] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[21] http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

where the evidence $Z = p(\mathbf{x})$ is a scaling factor dependent only on $x_1, \ldots, x_n$, that is, a constant if the values of the feature variables are known.

The NB Classifier This classifier pick the hypothesis that is most probable; this is known as the maximum a posteriori or MAP decision rule.

The corresponding classifier, a Bayes classifier, is the function that assigns a class label $\hat{y} = C_k$ for some k as follows:

$$\hat{y} = \underset{k \in \{1,\ldots,K\}}{argmax}\, p(C_k) \prod_{i=1}^{n} p(x_i|C_k)$$

Gaussian Naïve Bayes A typical assumption is that the values associated with each class are distributed according to a Gaussian distribution. For example, suppose the training data contain a continuous attribute, x. We first segment the data by the class, and then compute the mean and variance of x in each class. Let μ_c be the mean of the values in x associated with class c, and let σ_c^2 be the variance of the values in x associated with class c. Then, the probability distribution of some value given a class, $p(x = v|c)$, can be computed by plugging v into the equation for a Normal distribution parameterized by μ_c and σ_c^2. That is,

$$p(x = v|c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} e^{-\frac{(v-\mu_c)^2}{2\sigma_c^2}}$$

Bernoulli Naïve Bayes In the multivariate Bernoulli event model, features are independent booleans (binary variables) describing inputs. If x_i is a boolean expressing the occurrence or absence of the i'th term from the vocabulary, then the likelihood of a document given a class C_k is given by:

$p(\mathbf{x}|C_k) = \prod_{i=1}^{n} p_{ki}^{x_i}(1 - p_{ki})^{(1-x_i)}$

where p_{ki} is the probability of class C_k generating the term x_i.

Implementation The IDS Gaussian NB classifier was implemented using scikit-learn's sklearn.naive_bayes.GaussianNB class[22].

The IDS Bernoulli NB classifier was implemented using scikit-learn's sklearn.naive_bayes.BernoulliNB class[23].

5.4.5 AdaBoost Classifier AdaBoost, short for "Adaptive Boosting", can be used in conjunction with many other types of learning algorithms to improve their performance ([38]). The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. The core principle of AdaBoost is to fit a sequence of weak learners (i.e., models that are only slightly better than random guessing, such as small decision trees) on repeatedly modified versions of the data. The predictions from

[22] http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html
[23] http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html

all of them are then combined through a weighted majority vote (or sum) to produce the final prediction. The data modifications at each so-called boosting iteration consist of applying weights $w_1, w_2, ..., w_N$ to each of the training samples. Initially, those weights are all set to $w_i = 1/N$, so that the first step simply trains a weak learner on the original data. For each successive iteration, the sample weights are individually modified and the learning algorithm is reapplied to the reweighted data. At a given step, those training examples that were incorrectly predicted by the boosted model induced at the previous step have their weights increased, whereas the weights are decreased for those that were predicted correctly. As iterations proceed, examples that are difficult to predict receive ever-increasing influence. Each subsequent weak learner is thereby forced to concentrate on the examples that are missed by the previous ones in the sequence. AdaBoost (with decision trees as the weak learners) is often referred to as the best out-of-the-box classifier. Unlike neural networks and SVMs, the AdaBoost training process selects only those features known to improve the predictive power of the model, reducing dimensionality and potentially improving execution time as irrelevant features do not need to be computed.

Training AdaBoost refers to a particular method of training a boosted classifier. A boost classifier is a classifier in the form

$F_T(x) = \sum_{t=1}^{T} f_t(x)$

where each f_t is a weak learner that takes an object x as input and returns a real valued result indicating the class of the object. The sign of the weak learner output identifies the predicted object class and the absolute value gives the confidence in that classification. Similarly, the T-layer classifier will be positive if the sample is believed to be in the positive class and negative otherwise.

Each weak learner produces an output, hypothesis $h(x_i)$, for each sample in the training set. At each iteration t, a weak learner is selected and assigned a coefficient α_t such that the sum training error E_t of the resulting t-stage boost classifier is minimized.

$E_t = \sum_i E[F_{t-1}(x_i) + \alpha_t h(x_i)]$

Here $F_{t-1}(x)$ is the boosted classifier that has been built up to the previous stage of training, $E(F)$ is some error function and $f_t(x) = \alpha_t h(x)$ is the weak learner that is being considered for addition to the final classifier.

Weighting At each iteration of the training process, a weight is assigned to each sample in the training set equal to the current error $E(F_{t-1}(x_i))$ on that sample. These weights can be used to inform the training of the weak learner, for instance, decision trees can be grown that favor splitting sets of samples with high weights.

Discrete AdaBoost The discrete variant, used in our IDS, is as follows:

With:

- Samples $x_1 \dots x_n$
- Desired outputs $y_1 \dots y_n, y \in \{-1, 1\}$

– Initial weights $w_{1,1} \ldots w_{n,1}$ set to $\frac{1}{n}$
– Error function $E(f(x), y, i) = e^{-y_i f(x_i)}$
– Weak learners $h\colon x \to [-1, 1]$

For t in $1 \ldots T$:

1. Choose $f_t(x)$:

   (a) Find weak learner $h_t(x)$ that minimizes ϵ_t, the weighted sum error for misclassified points $\epsilon_t = \sum_i w_{i,t} E(h_t(x), y, i)$

   (b) Choose $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$

2. Add to ensemble:

   (a) $F_t(x) = F_{t-1}(x) + \alpha_t h_t(x)$

3. Update weights:

   (a) $w_{i,t+1} = w_{i,t} e^{-y_i \alpha_t h_t(x_i)}$for all i

   (b) Renormalize $w_{i,t+1}$ such that $\sum_i w_{i,t+1} = 1$

Implementation The IDS classifier was implemented using scikit-learn's sklearn.ensemble.AdaBoostClassifier class[24]. The default value is a maximum of 50 estimators for boosting (or less if a perfect fit is achieved before).

5.4.6 Support Vector Machine (SVM) Classifier A data point is viewed as a p-dimensional vector (a list of p numbers), and we want to know whether we can separate such points with a $(p-1)$-dimensional hyperplane. This is called a linear classifier. There are many hyperplanes that might classify the data. One reasonable choice as the best hyperplane is the one that represents the largest separation, or margin, between the two classes. So we choose the hyperplane so that the distance from it to the nearest data point on each side is maximized. If such a hyperplane exists, it is known as the maximum-margin hyperplane and the linear classifier it defines is known as a maximum margin classifier; or equivalently, the perceptron of optimal stability.

An example of such separation in Figure 5. H3 does not separate the classes. H1 does, but only with a small margin. H2 separates them with the maximum margin.

[24] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

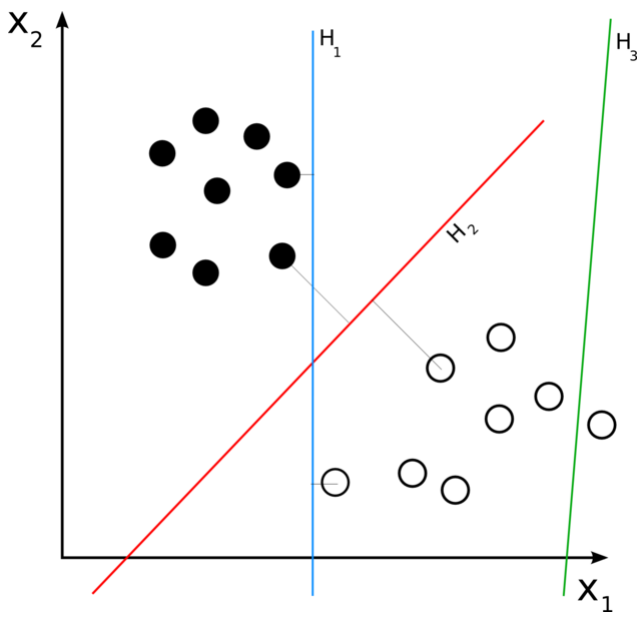


Fig. 5. Example of Separating Hyperplanes

Linear SVM Given some training data $\mathcal{D}$, a set of n points of the form

$\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid \mathbf{x}_i \in R^p, y_i \in \{-1, 1\}\}_{i=1}^n$

where the y_i is either 1 or -1, indicating the class to which the point $\mathbf{x}_i$ belongs ([40]). Each $\mathbf{x}_i$ is a p-dimensional real vector. We want to find the maximum-margin hyperplane that divides the points having $y_i = 1$ from those having $y_i = -1$. Any hyperplane can be written as the set of points $\mathbf{x}$ satisfying:

$\mathbf{w} \cdot \mathbf{x} - b = 0$

where $\cdot$ denotes the dot product and $\mathbf{w}$ the (not necessarily normalized) normal vector to the hyperplane. The parameter $\frac{b}{\|\mathbf{w}\|}$ determines the offset of the hyperplane from the origin along the normal vector $\mathbf{w}$.

If the training data are linearly separable, we can select two hyperplanes in a way that they separate the data and there are no points between them, and then try to maximize their distance. The region bounded by them is called "the margin". These hyperplanes can be described by the equations

$\mathbf{w} \cdot \mathbf{x} - b = 1$

and

$\mathbf{w} \cdot \mathbf{x} - b = -1$

By using geometry, we find the distance between these two hyperplanes is $\frac{2}{\|\mathbf{w}\|}$, so we want to minimize $\|\mathbf{w}\|$. As we also have to prevent data points from falling into the margin, we add the following constraint: for each i either

$\mathbf{w} \cdot \mathbf{x}_i - b \geq 1$ for $\mathbf{x}_i$of the first class

or

$\mathbf{w} \cdot \mathbf{x}_i - b \leq -1$ for $\mathbf{x}_i$of the second.

This can be rewritten as:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1, \text{ for all } 1 \leq i \leq n$$

We can put this together to get the optimization problem:

Minimize (in $\mathbf{w}, b$)

$$\|\mathbf{w}\|$$

subject to (for any $i = 1, \ldots, n$)

$$y_i(\mathbf{w} \cdot \mathbf{x_i} - b) \geq 1$$

In Figure 6 we see a maximum-margin hyperplane and margins for an SVM trained with samples from two classes. Samples on the margin are called the support vectors.

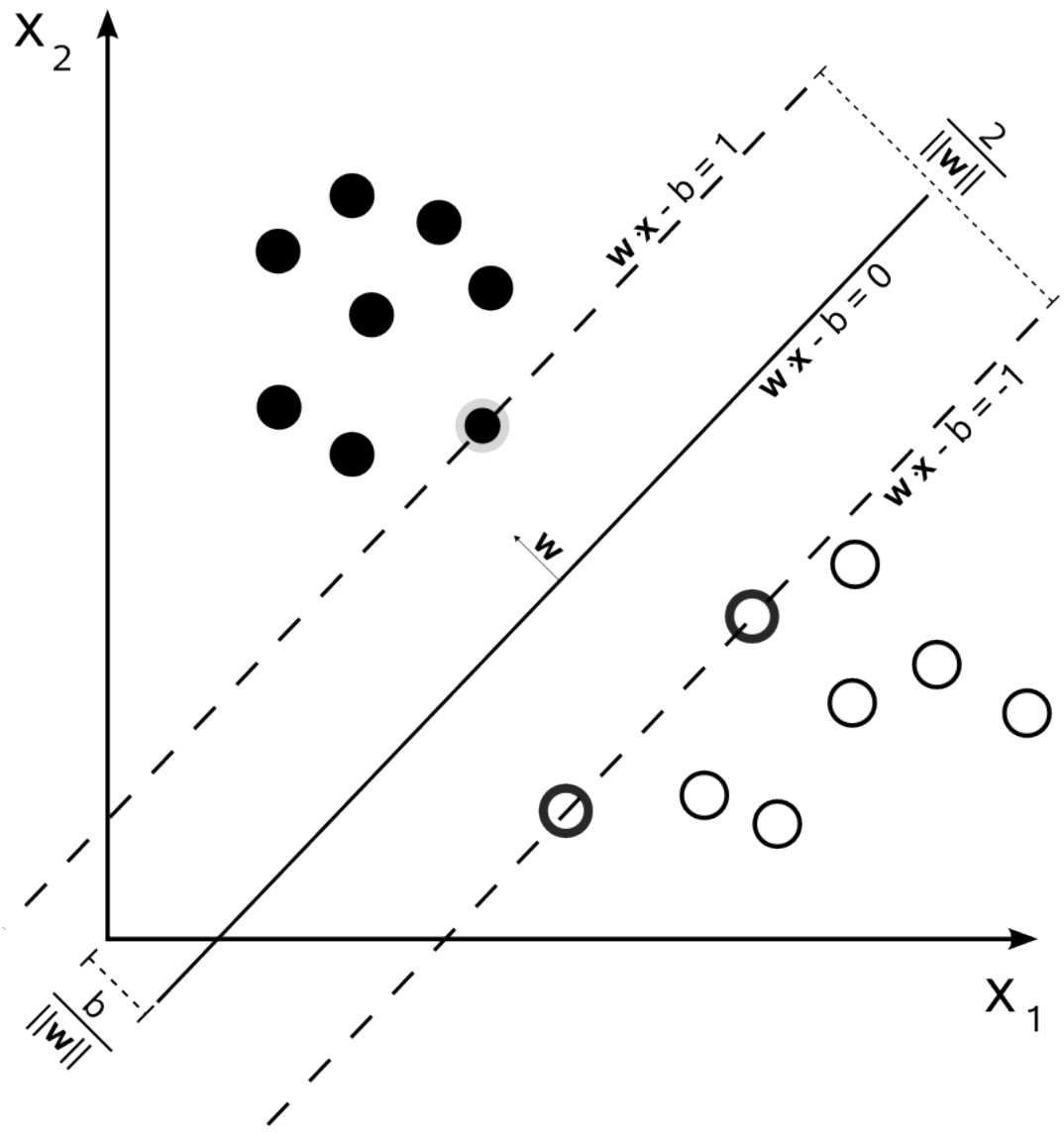


Fig. 6. Example of Separating Hyperplanes

Nonlinear classification A way to create nonlinear classifiers is by applying the kernel trick to maximum-margin hyperplanes ([34]). The resulting algorithm is formally similar, except that every dot product is replaced by a nonlinear kernel function. This allows the algorithm to fit the maximum-margin hyperplane in a transformed feature space. The transformation may be nonlinear and the transformed space high dimensional; thus though the classifier is a hyperplane

in the high-dimensional feature space, it may be nonlinear in the original input space.

An example of such a kernel function is a Gaussian Radial Basis Function (RBF):

$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|^2)$, for $\gamma > 0$. Sometimes parametrized using $\gamma = 1/2\sigma^2$

If the kernel used is a Gaussian radial basis function, the corresponding feature space is a Hilbert space of infinite dimensions.

he kernel is related to the transform $\varphi(\mathbf{x_i})$ by the equation $k(\mathbf{x_i}, \mathbf{x_j}) = \varphi(\mathbf{x_i}) \cdot \varphi(\mathbf{x_j})$. The value w is also in the transformed space, with $\mathbf{w} = \sum_i \alpha_i y_i \varphi(\mathbf{x}_i)$. Dot products with w for classification can again be computed by the kernel trick, i.e. $\mathbf{w} \cdot \varphi(\mathbf{x}) = \sum_i \alpha_i y_i k(\mathbf{x}_i, \mathbf{x})$

An example for the kernel trick is shown in Figure 7.

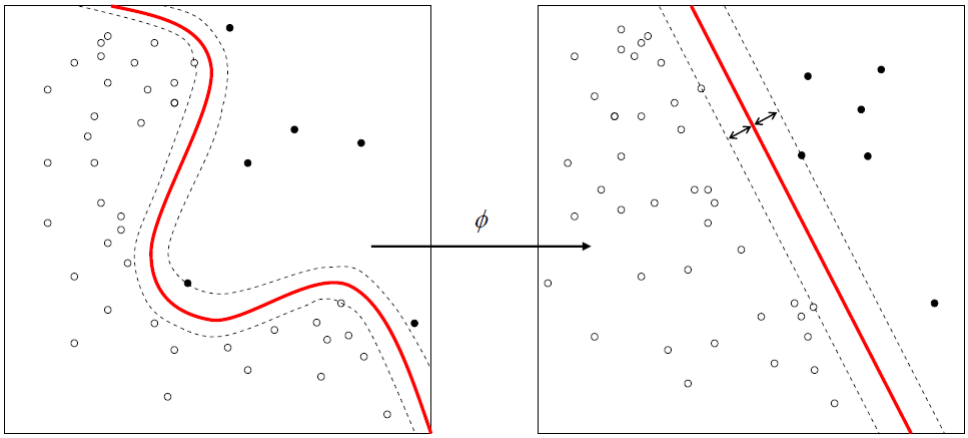


Fig. 7. Example of the Kernel Trick

Implementation The IDS Linear SVC classifier was implemented using scikit-learn's: sklearn.svm.LinearSVC class[25].

The IDS SVC classifier was implemented using scikit-learn's: sklearn.svm.SVC class[26], with a default RBF kernel.

5.4.7 Linear Discriminant Analysis (LDA) Classifier Linear discriminant analysis (LDA) is a generalization of Fisher's linear discriminant, a method used to find a linear combination of features that characterizes or separates two or more classes of objects or events. The resulting combination may be used as a linear classifier ([39]).

[25] http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
[26] http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

LDA for Two Classes The variants used here, for a binary classifier:

Consider a set of observations $\boldsymbol{x}$ (also called features, attributes, variables or measurements) for each sample of an object or event with known class y. This set of samples is called the training set. The classification problem is then to find a good predictor for the class y of any sample of the same distribution (not necessarily from the training set) given only an observation $\boldsymbol{x}$.

LDA approaches the problem by assuming that the conditional probability density functions $p(\boldsymbol{x}|y = 0)$ and $p(\boldsymbol{x}|y = 1)$ are both normally distributed with mean and covariance parameters $(\boldsymbol{\mu}_0, \Sigma_0)$ and$(\boldsymbol{\mu}_1, \Sigma_1)$, respectively. Under this assumption, the Bayes optimal solution is to predict points as being from the second class if the log of the likelihood ratios is below some threshold T, so that;

$(\boldsymbol{x} - \boldsymbol{\mu}_0)^T \Sigma_0^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_0) + \ln |\Sigma_0| - (\boldsymbol{x} - \boldsymbol{\mu}_1)^T \Sigma_1^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_1) - \ln |\Sigma_1| < T$

Without any further assumptions, the resulting classifier is referred to as QDA (quadratic discriminant analysis).

LDA instead makes the additional simplifying homoscedasticity assumption (i.e. that the class covariances are identical, so $\Sigma_0 = \Sigma_1 = \Sigma$) and that the covariances have full rank. In this case, several terms cancel:

$\boldsymbol{x}^T \Sigma_0^{-1} \boldsymbol{x} = \boldsymbol{x}^T \Sigma_1^{-1} \boldsymbol{x}$

$\boldsymbol{x}^T {\Sigma_i}^{-1} \boldsymbol{\mu_i} = \boldsymbol{\mu_i}^T {\Sigma_i}^{-1} \boldsymbol{x}$because Σ_i is Hermitian

and the above decision criterion becomes a threshold on the dot product

$\boldsymbol{w} \cdot \boldsymbol{x} > c$

for some threshold constant c, where

$\boldsymbol{w} \propto \Sigma^{-1} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)$

$c = \frac{1}{2} (T - \boldsymbol{\mu_0}^T \Sigma_0^{-1} \boldsymbol{\mu_0} + \boldsymbol{\mu_1}^T \Sigma_1^{-1} \boldsymbol{\mu_1})$

This means that the criterion of an input $\boldsymbol{x}$ being in a class y is purely a function of this linear combination of the known observations.

It is often useful to see this conclusion in geometrical terms: the criterion of an input $\boldsymbol{x}$ being in a class y is purely a function of projection of multidimensional-space point $\boldsymbol{x}$ onto vector $\boldsymbol{w}$ (thus, we only consider its direction). In other words, the observation belongs to y if corresponding $\boldsymbol{x}$ is located on a certain side of a hyperplane perpendicular to $\boldsymbol{w}$. The location of the plane is defined by the threshold c.

In practice, the class means and covariances are not known. They can, however, be estimated from the training set. Either the maximum likelihood estimate or the maximum a posteriori estimate may be used in place of the exact value in the above equations.

Implementation The IDS classifier was implemented using scikit-learn's: sklearn.lda.LDA class[27].

5.5 Wrapper

All the above parts have different interfaces and, as already mentioned, are inter-dependent from each-other. However, the user want a simple interface to classify

[27] http://scikit-learn.org/stable/modules/generated/sklearn.lda.LDA.html

a file, without dealing with issues such-as sand-boxing or feature selection. Thus, the *IDS* Python module is a Facade design pattern[28], which provides the user with a simplified, high-level interface and operates the other modules, as needed, without the user knowledge. The user knows only about this module and uses only its interface. It provides two external functions:

- The *IDS.__init__()* method is identical to the *classification.Predictor.__init__()* method. In-fact, the IDS class inherits from *classification.Predictor*. This was done in-order to be able to use *IDS* in any function that accepts a classifier (e.g., for plotting the IDS's decision tree internal classifier, etc.).
- The *IDS.inspect()* method gets a file path to inspect, calls *sandboxing.trace_proc_in_sandbox()* (see section 5.1) to record the system calls of it in a sandbox, calls *normalization.normalize()* (see section 5.3) on the output file to convert it to a unified XML format, calls *feature_parsing.parse_features()* (see section 5.3) on the XML file to convert it to a Python dictionary with the proper features and finally calls *IDS.predict()* (inherited from *Predictor.predict()*, shown in section 5.4) with this dictionary and returns the classification calculated by the classifier (after transforming the dictionary to a bit vector and performing feature selection): *1* for a malicious classification or *0* for a benign classification (and not a Boolean return value, to allow extensibility to the interface, e.g., returning the malware family and 0 for a benign file in the future.

6 Experimental Evaluation of the Basic IDS Model

In-order to test the true positive detection rate of our IDS for both benign software and malware, we've used benign files collection from the Program Files folder of Windows XP SP3 (Appendix B.1) and from our collection of third party benign programs (Appendix B.2) and malware from the type of Win32 Virus from VxHeaven collection (Appendix B.3). The test set size was 494 benign programs and 521 malware. The test set benign software and malware can be found in appendices B.4 and B.5, respectively. The files used to test the detection rate were not used to train the IDS and thus they present code that wasn't encountered before. The results are shown in table 1.

We can see that all classifiers out-performed random classification (with 50% detection rate for both malware and benign software).

We also see that, as expected, the Bernoulli NB classifier performs better than the Gaussian one, because the features are independent booleans.

Except for the NB classifiers (with both Gaussian and Bernoulli distributions) and the RBF-kernel based SVM, most classifiers had similar detection rates. The Random Forest classifier and k-Nearest Neighbors classifier were the best overall, taking into account both malware and benign software detection rate (by maximizing the MCC), where the k-Nearest Neighbors classifier was better in malware detection and the Random Forest classifier was better in benign software detection.

[28] https://en.wikipedia.org/wiki/Facade_pattern

Table 1. Detection Rate of the IDS by Classifier Type

Classifier Type	Malware Detection Rate (TPR)	Benign Software Detection Rate (TNR)	MCC
Decision Tree	83.37	90.49	0.755
Random Forest	86.07	89.47	0.770
K-Nearest Neighbors	89.36	86.03	0.770
Naïve Bayes (Gaussian)	87.04	54.45	0.503
Naïve Bayes (Bernoulli)	97.87	59.92	0.638
Ada-Boost	87.43	84.82	0.742
Support Vector Machine (Linear)	87.46	86.44	0.756
Support Vector Machine (RBF)	96.32	74.90	0.742
Linear Discriminant Analysis	82.59	82.59	0.682

7 Conclusions

In this paper, we have described the implementation of an IDS based on machine-learning classifier of various types, based on a dynamic analysis of the system calls executed by the inspected code, inside a sandbox, as features. We have shown that each of the classifiers' type out-performed random classification while coping with malicious and benign code which weren't encountered before. We see that Random Forest and k-NN classifiers gives us the best results for the training set used. One should note that the results are very close, and a different training and test sets could yield different results.

In our future work in this area, we would compare other classifiers (Artificial Neural Networks, etc.), which weren't implemented in the current IDS. We would also try to increase the detection rate of the IDS by increasing the training set and by applying input transformations to the classifier's input before using it.

References

1. Somayaji, A., Forrest, S.: Automated Response Using System-Call Delays. In: Proceedings of the 9th USENIX Security Symposium, pp. 185-198 (2000)
2. Warrender C., Forrest S., Pearlmutter B.: Detecting Intrusions Using System Calls: Alternative data models. In: Proceedings of the 1999 IEEE Symposium on Security and Privacy, pp. 133–145 (1999)
3. Forrest, S., Hofmeyr, S.A., Somayaji, A., Longsta, T.A.: A Sense of Self for Unix Processes. In: IEEE Symposium on Security and Privacy, pp. 120-128, IEEE Press, USA (1996)
4. Moskovitch, R., Gus, I., Pluderman, S., Stopel, D., Fermat, Y., Shahar, Y., Elovici, Y.: Host Based Intrusion Detection Using Machine Learning. In: Proceedings of Intelligence and Security Informatics, pp. 107-114 (2007)
5. Moskovitch R., Nissim N., Stopel D., Feher C., Englert R., Elovici Y.: Improving the Detection of Unknown Computer Worms Activity Using Active Learning. In: Proceedings of the 30th annual German conference on Advances in Artificial Intelligence, pp. 489 – 493 (2007)
6. Singhal, P., Raul, N.: Malware Detection Module Using Machine Learning Algorithms to Assist in Centralized Security in Enterprise Networks. In: International Journal of Network Security & Its Applications, Vol.4, No.1, pp. 661-67 (2012)
7. Rozenberg, B., Gudes, E., Elovici, Y., Fledel, Y.: Method for Detecting Unknown Malicious Executables. In: Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection, pp. 376-377 (2009)
8. Bayer, U., Kruegel, C., Kirda, E.: TTAnalyze: A Tool for Analyzing Malware. In: Proceedings of the 15th Ann. Conf. European Inst. for Computer Antivirus Research, pp. 180–192 (2006)
9. Willems, C., Holz, T., Freiling, F.: Toward Automated Dynamic Malware Analysis Using CWSandbox. IEEE: Security & Privacy, Volume 5 , Issue: 2, pp. 32 – 39 (2007)
10. Hunt, G.C., Brubacher, D.: Detours: Binary Interception of Win32 Functions. In: Proceedings of the 3rd Usenix Windows NT Symp., pp. 135–143 (1999)
11. Caruana, R., Niculescu-Mizil, A.: An Empirical Comparison of Supervised Learning Algorithms. ICML '06, Proceedings of the 23rd international conference on Machine learning, pp. 161-168.
12. Bellard F.: QEMU, A Fast and Portable Dynamic Translator. Proc. Usenix 2005 Ann. Technical Conf. (Usenix '05), Usenix Assoc., 2005, pp. 41–46.
13. Ferrie P.: Attacks on Virtual Machine Emulators. Symantec Advanced Threat Research.
14. Ferrie P.: Attacks on More Virtual Machine Emulators. Symantec Advanced Threat Research.
15. Raffetseder, T., Kruegel, C., Kirda, E.: Detecting System Emulators. In: Proceedings of Information Security, 10th International Conference, pp. 1-18 (2007)
16. Yin, H., Song, D., Egele, M., Kruegel, C., Kirda, E.: Panorama: Capturing System-wide Information Flow for Malware Detection and Analysis. In: Proceedings of the 14th ACM Conference of Computer and Communication Security, pp. 116-127 (2007)
17. King, S.T., Chen, P.M., Wang, Y.M., Verbowski, C., Wang, H.J., Lorch, J.R.: SubVirt: Implementing Malware with Virtual Machines. In: Proceedings of the 2006 IEEE Symposium on Security and Privacy, pp. 314– 327 (2006)
18. Fritsch H.: Analysis and Detection of Virtualization-Based Rootkits. TUM.

19. Moser, A., Kruegel, C., Kirda, E.: Limits of Static Analysis for Malware Detection. In: 23rd Annual Computer Security Applications Conference, pp. 421-430 (2007)
20. Santos, I., Brezo, F., Ugarte-Pedrero, X., Bringas, P.G.: Opcode sequences as representation of executables for data-mining-based unknown malware detection. In: Information Sciences 227, pp. 63–81 (2013)
21. Shabtai, A., Menahem, E., Elovici, Y.: F-Sign: Automatic, Function-Based Signature Generation for Malware. In: IEEE Trans. Systems, Man, and Cybernetics—Part C: Applications and Reviews, vol. 41, no. 4, pp. 494-508 (2011)
22. Quinlan, J.R.: Discovering rules from large collections of examples: A case study. In: Expert Systems in the Microelectronic Age, Michie, D. (Ed.), pp. 168-201 (1979)
23. Rutkowska, J., Tereshkin, A.: IsGameOver() Anyone?. Black Hat USA (2007)
24. Yang, Y., Fanglu, G., Susanta, N., Lap-chung, L., Tzi-cker, C.: A Feather-weight Virtual Machine for Windows Applications. In: Proceedings of the 2nd International Conference on Virtual Execution Environments, pp. 24-34 (2006)
25. Guyon, I., Elisseeff, A.: An introduction to Variable and Feature Selection. In: Journal of Machine Learning Research 3, pp. 1157-1182 (2003)
26. Zheng Z., Wu X., Srihari R.: Feature Selection for Text Categorization on Imbalanced Data. SIGKDD Explorations, 2002, pp. 80-89.
27. Yates, F.: Contingency table involving small numbers and the χ 2 test. In: Journal of the Royal Statistical Society, vol. 1-2, pp. 217-235 (1934)
28. Firdausi, I., Lim, C., Erwin, A.: Analysis of Machine Learning Techniques Used in Behavior Based Malware Detection. In: Proceedings of 2nd International Conference on Advances in Computing, Control and Telecommunication Technologies, pp. 201-203 (2010)
29. Kolter J.Z., Maloof M.A.: Learning to detect and classify malicious executables in the wild. In: Journal of Machine Learning Research, 7(Dec), pp. 2721 – 2744 (2006)
30. Kolter, J.Z., Maloof, M.A.: Learning to detect malicious executables in the wild. In: Proceedings of the 10th International Conference on Knowledge Discovery and Data Mining, pp. 470–478 (2004)
31. Schultz M., Eskin E., Zadok E., Stolfo S.: Data mining methods for detection of new malicious executables. In: Proceedings of the IEEE Symposium on Security and Privacy, pp. 38–49 (2001)
32. Tandon, G., Chan, P.: Learning Rules from System Call Arguments and Sequences for Anomaly Detection. In: ICDM Workshop on Data Mining for Computer Security, pp. 20-29, IEEE Press, USA (2003)
33. Egele, M., Scholte, T., Kirda, E., Kruegel, C.: A survey on automated dynamic malware-analysis techniques and tools. In: ACM Computing Surveys, Vol. 44, No. 2, Article 6, pp. 1-42 (2012)
34. Boser, B. E., Guyon, I. M., Vapnik, V. N.: A training algorithm for optimal margin classifiers. In: Proceedings of the fifth annual workshop on Computational learning theory - COLT '92, pp. 144 (1992)
35. Breiman, L.: Random Forests. In: Machine Learning 45 (1): pp. 5–32 (2001)
36. Hand, D. J., Yu, K.: Idiot's Bayes — not so stupid after all?. In: International Statistical Review 69 (3), pp. 385–399 (2001)
37. Altman, N. S.: An introduction to kernel and nearest-neighbor nonparametric regression. In: The American Statistician 46 (3), pp. 175–185 (1992)
38. Freund Y., Schapire R. E.: A decision-theoretic generalization of on-line learning and an application to boosting. In: Journal of Computer and System Sciences, 55(1), pp. 119–139 (1997)

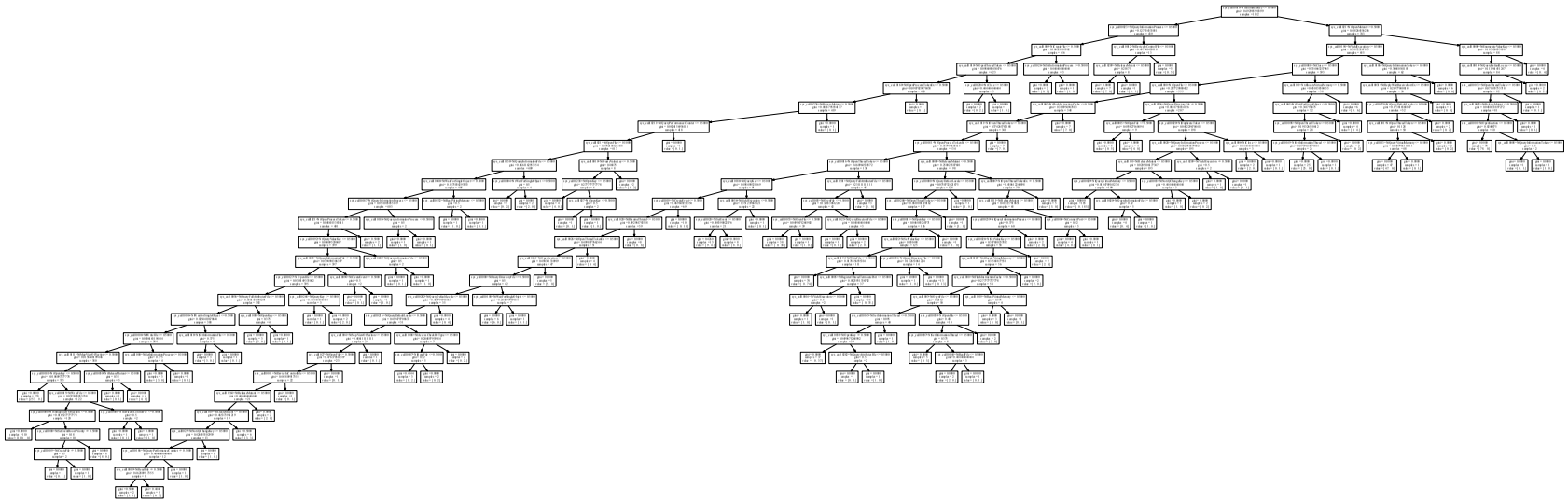
39. Swets D.L., Weng J.J.: Using Discriminant Eigenfeatures for Image Retrieval. In: IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 18, no. 8, pp. 831-836 (1996)
40. Vapnik V., Lerner A.: Pattern recognition using generalized portrait method. In: Automation and Remote Control, 24, pp. 774–780 (1963)
41. Powers D.: Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation. In: Journal of Machine Learning Technologies 2 (1): 37–63 (2011)
42. Gamma E., Helm R., Johnson R., Vlissides J.: Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA (1995)
43. Hoglund G., Butler J.: Rootkits: Subverting the Windows Kernel. Addison-Wesley (2005)

A Appendix A: The decision tree used by the IDS classifier

Attached is the decision tree created from the training set.

If the decision tree condition specified in a node is, e.g.: sys_call10018=NtEnumerateKey, this means the right child of this node in the decision path is chosen if *sys_call[10018]=NtEnumerateKey,* and the left chils is chosen if not.

A leaf node has a value vector of *[benign_training_set_samples, malicious_training_set_samples]*. Thus, an inspected code would be classified as malicious if —*malicious_training_set_samples*— ¿ —*benign_training_set_samples*— at that node, or as benign otherwise.



B Appendix B: Lists of Programs and Viruses Used by the IDS

B.1 System32 Benign Programs List (Original DB)

1. accwiz.exe
2. actmovie.exe
3. agentsvr.exe
4. ahui.exe
5. append.exe
6. arp.exe
7. asr_fmt.exe
8. asr_ldm.exe
9. asr_pfu.exe
10. atmadm.exe
11. attrib.exe
12. auditusr.exe
13. author.exe
14. bckgzm.exe
15. blastcln.exe
16. bootcfg.exe
17. bootok.exe
18. bootvrfy.exe
19. cacls.exe
20. calc.exe
21. cb32.exe
22. cfgwiz.exe
23. change.exe
24. charmap.exe
25. chglogon.exe
26. chgport.exe
27. chgusr.exe
28. chkdsk.exe
29. chkntfs.exe
30. chkrzm.exe
31. cidaemon.exe
32. cintsetp.exe
33. cisvc.exe
34. ckcnv.exe
35. cleanmgr.exe
36. cliconfg.exe
37. clipbrd.exe
38. clipsrv.exe
39. cmd.exe
40. cmdl32.exe
41. cmmon32.exe

42. cmstp.exe
43. comp.exe
44. compact.exe
45. comrepl.exe
46. comrereg.exe
47. conf.exe
48. conime.exe
49. control.exe
50. convert.exe
51. convlog.exe
52. cplexe.exe
53. cprofile.exe
54. cscript.exe
55. ctfmon.exe
56. davcdata.exe
57. dcomcnfg.exe
58. ddeshare.exe
59. debug.exe
60. defrag.exe
61. dfrgfat.exe
62. dfrgntfs.exe
63. diantz.exe
64. diskpart.exe
65. diskperf.exe
66. dllhst3g.exe
67. dmremote.exe
68. doskey.exe
69. dosx.exe
70. dplaysvr.exe
71. dpnsvr.exe
72. dpvsetup.exe
73. drvqry.exe
74. drwatson.exe
75. drwtsn32.exe
76. dumprep.exe
77. dvdupgrd.exe
78. dwwin.exe
79. dxdiag.exe
80. edlin.exe
81. esentutl.exe
82. eudcedit.exe
83. evcreate.exe
84. eventvwr.exe
85. evntcmd.exe
86. evntwin.exe
87. evtrig.exe

88. EXCH_regtrace.exe
89. exe2bin.exe
90. expand.exe
91. explorer.exe
92. extrac32.exe
93. fastopen.exe
94. fc.exe
95. find.exe
96. findstr.exe
97. finger.exe
98. fixmapi.exe
99. flattemp.exe
100. fontview.exe
101. forcedos.exe
102. fp98sadm.exe
103. fp98swin.exe
104. fpadmcgi.exe
105. fpcount.exe
106. fpremadm.exe
107. freecell.exe
108. fsutil.exe
109. ftp.exe
110. fxsclnt.exe
111. fxscover.exe
112. fxssend.exe
113. fxssvc.exe
114. gdi.exe
115. getmac.exe
116. gprslt.exe
117. gpupdate.exe
118. grpconv.exe
119. help.exe
120. helpctr.exe
121. helphost.exe
122. helpsvc.exe
123. hh.exe
124. hostname.exe
125. hrtzzm.exe
126. hscupd.exe
127. icwconn1.exe
128. icwconn2.exe
129. icwrmind.exe
130. icwtutor.exe
131. ie4uinit.exe
132. iedw.exe
133. iexplore.exe

134. iexpress.exe
135. iisreset.exe
136. iisrstas.exe
137. iissync.exe
138. imapi.exe
139. imekrmig.exe
140. imepadsv.exe
141. imjpdadm.exe
142. imjpdct.exe
143. imjpdsvr.exe
144. imjpinst.exe
145. imjpmig.exe
146. imjprw.exe
147. imjpuex.exe
148. imjputy.exe
149. imkrinst.exe
150. imscinst.exe
151. inetin51.exe
152. inetmgr.exe
153. inetwiz.exe
154. ipconfig.exe
155. ipsec6.exe
156. ipv6.exe
157. ipxroute.exe
158. isignup.exe
159. krnl386.exe
160. label.exe
161. lhmstsc.exe
162. lights.exe
163. lnkstub.exe
164. locator.exe
165. lodctr.exe
166. logagent.exe
167. logman.exe

B.2 Third-Party Benign Programs List (Original DB)

1. $0.exe
2. _007_PresentationHost_X86.exe
3. _MSRSTRT.EXE.exe
4. 3DVision_285.62.exe
5. 50comupd.exe
6. 7zFMn.exe
7. 7zgn.exe
8. aaw2007_13.exe
9. acdsee-15.exe

10. acdsee2009-11-0-113-en.exe
11. Ad-AwareAE.exe
12. adxregistrator.exe
13. aeldr.exe
14. AIMLang.exe
15. Alcohol.exe
16. ALUSDSVC.EXE.exe
17. AppleMobileDeviceHelper.exe
18. AppUpgrade.exe
19. ASBarBroker.exe
20. asvc.exe
21. avc-free_25.exe
22. avc-free_8.exe
23. avc-free_85.exe
24. avgscanx.exe
25. avgsystx.exe
26. avhlp.exe
27. avidemux.exe
28. avidemux_jobs.exe
29. AVMCDLG.exe
30. avnotify.exe
31. AVNOTIFY.EXE.exe
32. avrestart.exe
33. AWSC.exe
34. AxAutoMntSrv.exe
35. AxCmd.exe
36. Azureus_3_0_windows.exe
37. BackItUp.exe
38. backup.exe
39. BANG.EXE.exe
40. BCompare-3.2.3.13046.exe
41. BCompare-3.3.8.16340.exe
42. beta_6.exe
43. Binaries.exe
44. Binary.InstUpdate.exe
45. BitComet.exe
46. BitComet.RegTool.$[52].exe
47. BitTorrent.exe
48. BitTorrent_16.exe
49. BitTorrent_23.exe
50. BitTorrent_33.exe
51. BitTorrent_34.exe
52. BitTorrent_9.exe
53. Boomerang.exe
54. bootexctrl.exe
55. btwizard.exe

56. BugReport.exe
57. bunzip2.exe
58. cal.exe
59. ccIMScn.exe
60. CCleaner.exe
61. ccSetMgr.exe
62. ChangeIcon.exe
63. CheatEngine62_2.exe
64. chrome.exe
65. CLDetect.exe
66. cleaner.exe
67. codecmanager.exe
68. CometBrowser.exe
69. comm.exe
70. compress.exe
71. conf.exe
72. config.exe
73. corecmd.exe
74. Corel_PaintShopPro1000_EN_TBYB_TrialESD.exe
75. CoverDesC69A8C2C.exe
76. CrashReport.exe
77. CS_Dv32_5.exe
78. CWShredder_3.exe
79. DeepBurner.exe
80. default.exe
81. defaults.exe
82. devcon.exe
83. df.exe
84. digsby.exe
85. dirname.exe
86. DiscSpeed56258BEC.exe
87. disthelper.exe
88. dMC-R14-Ref-Trial.exe
89. dotmacsyncclient.exe
90. dotNetFx40_Client_x86_x64.exe
91. DPLaunch.exe
92. dpnsvr.exe
93. dtexec.exe
94. DW20.EXE.exe
95. DX9NT.exe
96. dxdiag.exe
97. editcap.exe
98. emoticonSelector.exe
99. emule.exe
100. epm_9.exe
101. ethereal.exe

102. Evernote_3.5.2.1663_2.exe
103. Evernote_3.5.6.2848.exe
104. extrac32.exe
105. eye-candy-7.1.0.1184_2.exe
106. F.bin.myisamlog.exe
107. F.bin.mysqlcheck.exe
108. F.bin.mysqlimport.exe
109. F.bin.mysqlshow.exe
110. fact.exe
111. fd35beta_2.exe
112. fd35beta_5.exe
113. fd35beta_6.exe
114. fd4beta_3.exe
115. FFPage.exe
116. FGOpenHelpOption.exe
117. file__x86__StartW8Button.exe
118. file__x86__StartW8Service.exe
119. FileZilla.exe
120. flash.exe
121. Flashget3.exe
122. FlashGetOpenHelp.exe
123. fold.exe
124. FPVGettingStarted.exe
125. FreemakeVideoConverter_2.1.2.0.exe
126. FreemakeVideoConverter_2.1.2.1.exe
127. FreemakeVideoConverter_2.3.2.0_2.exe
128. FreemakeVideoConverter_2.4.0.2.exe
129. FreemakeVideoConverter_2.4.0.8.exe
130. FreemakeVideoConverter_3.0.1.1_2.exe
131. FreemakeVideoConverter_3.0.1.10_2.exe
132. FreemakeVideoConverter_3.0.1.3.exe
133. FreemakeVideoConverter_3.0.2.15_2.exe
134. FreemakeVideoConverter_3.1.2.0.exe
135. FreemakeVideoConverter_4.0.0.5.exe
136. FreemakeVideoConverter_4.0.1.3_2.exe
137. FreemakeVideoConverter_4.0.1.5.exe
138. FreemakeVideoConverter_4.0.2.13.exe
139. FreemakeVideoConverter_4.0.2.14.exe
140. FreemakeVideoConverter_4.0.2.18_2.exe
141. FreemakeVideoConverter_4.0.2.2.exe
142. gdpluginregister.exe
143. GenerateJavaInterfaces.exe
144. GenPat.exe
145. GenRNKey.exe
146. Gizmo5.exe
147. GOM.EXE.exe

148. Google_Earth_EARG_opt_ff2_en-US.exe
149. GoogleEarth4.2.180.1134.exe
150. GoogleEarth4.2.181.2634.exe
151. GoogleEarthWin_31.exe
152. GoogleSketchUpWEN_10.exe
153. GoogleSketchUpWEN_14.exe
154. googletalk.exe
155. GPU-Z.0.2.2_2.exe
156. GPU-Z.0.2.8.exe
157. GPU-Z.0.3.0.exe
158. GPU-Z.0.3.2.exe
159. GPU-Z.0.4.4.exe
160. GPU-Z.0.5.0.exe
161. GPU-Z.0.5.2.exe
162. GPU-Z.0.5.5.exe
163. GPU-Z.0.5.8.exe
164. GPU-Z.0.6.4.exe
165. GPU-Z.0.6.9.exe
166. gspawn-win32-helper.exe
167. gt.exe
168. GTGCAPI.exe
169. guard.exe
170. GUARDGUI.EXE.exe
171. Handbrake.exe
172. head.exe
173. Icon._0303ABFCEA91CC458E85F5.exe
174. Icon._1184C67D76B1A4F6A92566.exe
175. Icon._156D70CAFB2BEE2ABE5748.exe
176. Icon._1FB51FDFCFA205F65BC588.exe
177. Icon._21F3885A18D238E15AAE81.exe
178. Icon._2314181B58997B6BCD7B79.exe
179. Icon._35BD4E88C121C15C5E2BF7.exe
180. Icon._59B301794351CEEAEF58DF.exe
181. Icon._6117BD307EC3285D33AFDB.exe
182. Icon._6FB838DCF5B179E225A45C.exe
183. Icon._6FEFF9B68218417F98F549.exe
184. Icon._768AB02C92FD35ABB7BD7B.exe
185. Icon._7DC9B802A0B4DFC670F340.exe
186. Icon._873F20797E2368199D2062.exe
187. Icon._9AFF48FAF9D2857DECD996.exe
188. Icon._A0EF104B0724158C88DE82.exe
189. Icon._AA3D3FDCACE452535CE053.exe
190. Icon._ADB6F693FF863D7D745BDE.exe
191. Icon._B31CA4D2B598F948A2C35E.exe
192. Icon._BEFA7CFB89910355116CC4.exe
193. Icon._C6DAF7E354977BBC39A2B1.exe

194. Icon._D263814FC8E1FA49FCA3F9.exe
195. Icon._D707CE1C009F1381803C2C.exe
196. Icon._DC0818A5B4E79AC3E76C6F.exe
197. Icon._E91286A1D4687069204F8C.exe
198. Icon._FB7C2905E8DA11F2EBCB4B.exe
199. Icon._FC68627DD502D4637AC328.exe
200. Icon.eset.exe
201. Icon.ico_kss.1.0.0.468.ico.exe
202. Icon.MsblIco.Exe.exe
203. Icon.POWERARC.exe
204. Icon.scalc.exe
205. Icon.ShortcutOGL_EB071909B9884F8CBF3D6115D4ADEE5E.exe
206. Icon.soffice.exe
207. iconworkshop_13.exe
208. iconworkshop_15.exe
209. iconworkshop_6.exe
210. id.exe
211. idman612_3.exe
212. idman612_4.exe
213. idman615.exe
214. idman615_2.exe
215. idman615_6.exe
216. idman615_9.exe
217. idman616_2.exe
218. idman617.exe
219. idman617b2.exe
220. idman617b3.exe
221. idman617b6.exe
222. idman617b7.exe
223. IdsInst.exe
224. IM7z.exe
225. iMeshV6_4.exe
226. ImportDS.exe
227. InCDE744FC5C.exe
228. InCDsv9xFF985424.exe
229. insthlp.exe
230. ipdrvupd.exe
231. isobuster_all_lang_14.exe
232. isobuster_all_lang_15.exe
233. isobuster_all_lang_16.exe
234. isobuster_all_lang_17.exe
235. isobuster_all_lang_25.exe
236. iview399.exe
237. JkDefrag.exe
238. JkDefragCmd.exe
239. jqsnotify.exe

240. jureg.exe
241. jusched.exe
242. kerio.exe
243. kerio-kpf-4.2.1-896-win_2.exe
244. KindleForPC.exe
245. lame.exe
246. Languages.exe
247. LanguageSelector.exe
248. lesskey.exe
249. LicenseActivator.exe
250. loader.exe
251. LogitechUpdate.exe
252. ls.exe
253. LSPVIEW.EXE.exe
254. LSUpdateManager.exe
255. LuAll.exe
256. LUCheck.EXE.exe
257. LUInit.exe
258. m1s_cn.exe
259. MakeLangId.exe
260. MapCalibrator.exe
261. mbsa.exe
262. md5sum.exe
263. MediaEspresso.exe
264. MediaMonkey_3.0.2.1134.exe
265. MediaMonkey_3.1.0.1201_Debug.exe
266. MediaMonkey_3.1.0.1213_Debug.exe
267. MediaMonkey_3.1.2.1267.exe
268. MediaMonkey_3.2.2.1299.exe
269. MediaMonkey_4.0.3.1472.exe
270. MediaMonkey_4.0.5.1494.exe
271. MediaPortal.DeployTool.exe
272. mirc719.exe
273. modern.exe
274. msgr7us_2.exe
275. msi_pkgchk.exe
276. msi-pkgchk.exe
277. navigator.exe
278. NBSFtpD4BC8A15.exe
279. nconvert.exe
280. ndiff.exe
281. Nero-11.2.00900_trial.exe
282. NeroCmd6A705341.exe
283. NeroGadgetCMServer31BFBA71.exe
284. NeroPatentActivationEAC2FF7A.exe
285. NeroRemoteCtrlHandlerF5088011.exe

286. NMain.exe
287. nmap_service.exe
288. nod32.exe
289. nping.exe
290. NSIS.Library.RegTool.exe
291. NSIS.Library.RegTool.v2.$[38].exe
292. NSIS.Library.RegTool.v2.$[70].exe
293. NSIS.Library.RegTool.v3.$[71].exe
294. nsplugin.exe
295. nvSmartMaxapp.exe
296. NvStereoUtilityOGL.exe
297. nvStInst.exe
298. nvStReg.exe
299. NvStTest.Exe.exe
300. oax0i8iu.exe
301. objectdock_freeware_13.exe
302. objectdock_freeware_9.exe
303. ocpinst.exe
304. OrbitDownloader_4.1.1.17.exe
305. PaintShopPro1111_EN_DE_FR_ES_IT_NL_CORELTBYB_ESD.exe
306. pango-querymodules.exe
307. partinfo.exe
308. paste.exe
309. patch.exe
310. patchbeam10001.exe
311. PhraseExpress.exe
312. PhraseExpress_3.exe
313. PicasaMediaDetector.exe
314. PicasaRestore.exe
315. PictureViewer.exe
316. point32.exe
317. postgresql-9.1.3-1-windows.exe
318. postgresql-9.1.5-1-windows_2.exe
319. postgresql-9.2.0-1-windows.exe
320. powarc960_3.exe
321. powarc961.exe
322. PowerDVD12.exe
323. PowerDVD12Agent.exe
324. PRFA-IEToolbar.exe
325. PrimoRun.exe
326. ProInfoPack.exe
327. pxhpinst.exe
328. QMPlayer.exe
329. qtconfig.exe
330. QTInfo.exe
331. rdsconfig.exe

332. RealPlayer_10.exe
333. RealPlayer_12.exe
334. RealPlayer_15.exe
335. RealPlayer_7.exe
336. redit.exe
337. Reflect.exe
338. regcomp.exe
339. regRD.exe
340. regsvr32.exe
341. Remover.exe
342. RMEncoder.exe
343. RootkitRevealer.exe
344. rpcapd.exe
345. san1230.exe
346. san15124.exe
347. san1610.exe
348. san1611.exe
349. san1720.exe
350. san1747.exe
351. san1820.exe
352. san1824.exe
353. san1830.exe
354. san1852_2.exe
355. san1950_2.exe
356. san2011-1759.exe
357. sbAutoPlayUtil.exe
358. sc14_2.exe
359. scalc.exe
360. scenarioengine.exe
361. seamonkey.exe
362. setreg.exe
363. slide.exe
364. smplayer.exe
365. smtube.exe
366. snagit.exe
367. snagit_3.exe
368. soffice.exe
369. songbird.exe
370. spf.exe
371. SpiderOak.exe
372. spybot-2.1.exe
373. spybotsd152.exe
374. spybotsd160-beta1.exe
375. spybotsd162.exe
376. SpywareTerminator.exe
377. SpywareTerminator_12.exe

378. SpywareTerminator_4.exe
379. SpywareTerminator_6.exe
380. sqlsqm.exe
381. SSAutoRN.exe
382. sslisten.exe
383. stclient_wrapper.exe
384. Stinger.exe
385. stinger32_23.exe
386. stinger32_27.exe
387. stinger32_35.exe
388. stinger32_37.exe
389. stinger32_46.exe
390. stinger32_51.exe
391. StreamServer.exe
392. stub.exe
393. Stub.exe
394. SumatraPDF.exe
395. Sunbelt-Personal-Firewall_7.exe
396. SUPERAntiSpyware_101.exe
397. SUPERAntiSpyware_28.exe
398. SUPERAntiSpyware_37.exe
399. SUPERAntiSpyware_63.exe
400. SUPERAntiSpyware_66.exe
401. SUPERAntiSpyware_7.exe
402. SUPERAntiSpyware_81.exe
403. SWHELPER.EXE.exe
404. SWHELPER_1020022.EXE.exe
405. SwHelper_1165635.exe
406. SymLnch.exe
407. TagRename365.exe
408. TagRename366.exe
409. teracopy227.exe
410. text2pcap.exe
411. tnameserv.exe
412. toolbar.exe
413. TopStyle50_12.exe
414. TopStyle50_5.exe
415. TopStyle50_8.exe
416. TopStyle50_9.exe
417. udefrag-gui-config.exe
418. UNINST.EXE.exe
419. UninstPCS.exe
420. UninstPCSFEMsi.exe
421. UNNeroBackItUp98571DFD.exe
422. unopkg.exe
423. unregmp2.exe

424. UNWISE32.EXE.exe
425. updateLists.exe
426. utorrent-1.5.1-beta-build-463.exe
427. utorrent-1.6-beta-build-467_2.exe
428. verse.exe
429. VideoSnapshot.exe
430. VirtualBox-2.2.0-45846-Win.exe
431. VirtualBox-2.2.2-46594-Win.exe
432. VirtualBox-3.1.0-55467-Win.exe
433. VirtualBox-3.1.2-56127-Win.exe
434. VirtualBox-3.2.0-61806-Win.exe
435. VirtualBox-3.2.8-64453-Win.exe
436. VirtualBox-4.0.0-69151-Win.exe
437. VistaCodecs.exe
438. VMware-player-3.1.1-282343.exe
439. vnc-4_1_2-x86_win32.exe
440. VNC-5.0.0-Windows_2.exe
441. VNC-5.0.4-Windows_2.exe
442. VProTray.exe
443. Vuze_4.0.0.2_windows.exe
444. Vuze_4.2.0.4_windows.exe
445. Vuze_4.2.0.8_windows.exe
446. Vuze_4.3.1.0a_windows_2.exe
447. Vuze_4402_windows.exe
448. Vuze_4502_windows.exe
449. Vuze_4502b_windows.exe
450. Vuze_4510b_windows.exe
451. Vuze_4604_windows.exe
452. vwpt.exe
453. WC32TO16.EXE.exe
454. widgetsus_14.exe
455. windowblinds4_public.exe
456. windowblinds5_public_3.exe
457. WindowBlinds6_public_3.exe
458. WindowBlinds6_public_4.exe
459. WindowBlinds7_public_2.exe
460. WindowBlinds7_public_4.exe
461. WindowBlinds7_public_5.exe
462. windows_dir_watcher.exe
463. WindowsXP-KB942288-v3-x86.exe
464. WindowsXP-MSCompPackV1-x86.exe
465. wininst_10.0.exe
466. wininst_7.1.exe
467. wininst-6.exe
468. wininst-9.0.exe
469. WinPcap_3_1.exe

470. WinRAR.exe
471. wlunpacker.exe
472. wmfdist95.exe
473. wmp11.exe
474. WMPBurn.exe
475. WMPBurn9F9ABA59.exe
476. word-list-compress.exe
477. wsctool.exe
478. wudfhost.exe
479. WzPreviewer32.exe
480. WZQKPICK.EXE.exe
481. WZSESS32.EXE.exe
482. YahooWidgets.exe
483. yandex_downloader.exe
484. ymsgr702_120_us.exe
485. ymsgr750_333_us.exe
486. ymsgr7us.exe
487. yset.exe
488. YTB.exe
489. ZATRAY.EXE.exe
490. ZipSendService.exe
491. ZLUNWISE.EXE.exe
492. ZONESTUB.EXE.exe
493. zplayer.exe
494. zpupdate.exe

B.3 Malicious Programs List (Original DB)

1. Virus.Win32.Adson.1703.exe
2. Virus.Win32.Afgan.e.exe
3. Virus.Win32.Agent.ak.exe
4. Virus.Win32.Agent.bs.exe
5. Virus.Win32.Agent.ce.exe
6. Virus.Win32.Agent.cj.exe
7. Virus.Win32.Aidlot.exe
8. Virus.Win32.Alcaul.e.exe
9. Virus.Win32.Alcaul.i.exe
10. Virus.Win32.Alcaul.m.exe
11. Virus.Win32.Aldebaran.8365.b.exe
12. Virus.Win32.Aliser.8381.exe
13. Virus.Win32.Alma.5319.exe
14. Virus.Win32.Anuir.3746.exe
15. Virus.Win32.Anuir.a.exe
16. Virus.Win32.AOC.3649.b.exe
17. Virus.Win32.Apathy.5378.exe
18. Virus.Win32.Arcer.exe

19. Virus.Win32.Arrow.a.exe
20. Virus.Win32.Asorl.b.exe
21. Virus.Win32.AutoIt.j.exe
22. Virus.Win32.AutoRun.aku.exe
23. Virus.Win32.Autorun.fr.exe
24. Virus.Win32.Awfull.3318.exe
25. Virus.Win32.Baklajan.a.exe
26. Virus.Win32.Basket.a.exe
27. Virus.Win32.Beef.2110.exe
28. Virus.Win32.Belial.b.exe
29. Virus.Win32.Belial.f.exe
30. Virus.Win32.Benny.3219.a.exe
31. Virus.Win32.Bika.1857.exe
32. Virus.Win32.Blakan.2016.exe
33. Virus.Win32.Blakan.exe
34. Virus.Win32.Bluwin.a.exe
35. Virus.Win32.Bolzano.2664.exe
36. Virus.Win32.Bolzano.3120.exe
37. Virus.Win32.Bolzano.3628.exe
38. Virus.Win32.Bolzano.4096.c.exe
39. Virus.Win32.Bolzano.5396.a.exe
40. Virus.Win32.Bonding.a.exe
41. Virus.Win32.Bube.b.exe
42. Virus.Win32.Bube.g.exe
43. Virus.Win32.Bube.l.exe
44. Virus.Win32.Butter.exe
45. Virus.Win32.Cabanas.b.exe
46. Virus.Win32.Cabanas.MsgBox.exe
47. Virus.Win32.Cargo.11935.exe
48. Virus.Win32.Ceel.b.exe
49. Virus.Win32.Champ.5464.exe
50. Virus.Win32.Champ.5536.exe
51. Virus.Win32.Champ.5722.exe
52. Virus.Win32.Cheburgen.a.exe
53. Virus.Win32.Chiton.d.exe
54. Virus.Win32.Chiton.h.exe
55. Virus.Win32.Chiton.o.exe
56. Virus.Win32.Chiton.t.exe
57. Virus.Win32.Chuzy.a.exe
58. Virus.Win32.Cloz.a.exe
59. Virus.Win32.Cornad.exe
60. Virus.Win32.Crunk.a.exe
61. Virus.Win32.Crypto.c.exe
62. Virus.Win32.CTZA.a.exe
63. Virus.Win32.Damm.1537.b.exe
64. Virus.Win32.Damm.1647.b.exe

65. Virus.Win32.Datus.exe
66. Virus.Win32.Deemo.3028.exe
67. Virus.Win32.Delf.ad.exe
68. Virus.Win32.Delf.aj.exe
69. Virus.Win32.Delf.ar.exe
70. Virus.Win32.Delf.b.exe
71. Virus.Win32.Delf.bg.exe
72. Virus.Win32.Delf.bk.exe
73. Virus.Win32.Delf.br.exe
74. Virus.Win32.Delf.bw.exe
75. Virus.Win32.Delf.cb.exe
76. Virus.Win32.Delf.cl.exe
77. Virus.Win32.Delf.cs.exe
78. Virus.Win32.Delf.cx.exe
79. Virus.Win32.Delf.dc.exe
80. Virus.Win32.Delf.i.exe
81. Virus.Win32.Delf.n.exe
82. Virus.Win32.Delf.s.exe
83. Virus.Win32.Delfer.a.exe
84. Virus.Win32.Dias.b.exe
85. Virus.Win32.Diehard.a.exe
86. Virus.Win32.Ditex.a.exe
87. Virus.Win32.Dobom.a.exe
88. Virus.Win32.Donut.exe
89. Virus.Win32.Doser.4190.exe
90. Virus.Win32.Doser.4540.exe
91. Virus.Win32.Downloader.ab.exe
92. Virus.Win32.Downloader.ag.exe
93. Virus.Win32.Downloader.al.exe
94. Virus.Win32.Downloader.ap.exe
95. Virus.Win32.Downloader.au.exe
96. Virus.Win32.Downloader.ay.exe
97. Virus.Win32.Downloader.bd.exe
98. Virus.Win32.Downloader.bi.exe
99. Virus.Win32.Downloader.d.exe
100. Virus.Win32.Downloader.k.exe
101. Virus.Win32.Downloader.o.exe
102. Virus.Win32.Downloader.u.exe
103. Virus.Win32.Downloader.z.exe
104. Virus.Win32.Drivalon.2148.exe
105. Virus.Win32.Dropet.790.exe
106. Virus.Win32.Drowor.g.exe
107. Virus.Win32.Dzan.a.exe
108. Virus.Win32.Eclipse.c.exe
109. Virus.Win32.Egolet.c.exe
110. Virus.Win32.Elkern.b.exe

111. Virus.Win32.Emar.a.exe
112. Virus.Win32.Emotion.d.exe
113. Virus.Win32.Enerlam.b.exe
114. Virus.Win32.Enumiacs.6656.exe
115. Virus.Win32.Etap.exe
116. Virus.Win32.Eva.d.exe
117. Virus.Win32.Evar.3582.exe
118. Virus.Win32.Evol.c.exe
119. Virus.Win32.Evul.8192.d.exe
120. Virus.Win32.Evul.8192.h.exe
121. Virus.Win32.Evyl.g.exe
122. Virus.Win32.Expiro.c.exe
123. Virus.Win32.Expiro.g.exe
124. Virus.Win32.Expiro.l.exe
125. Virus.Win32.Expiro.p.exe
126. Virus.Win32.Fighter.a.exe
127. Virus.Win32.Flopex.a.exe
128. Virus.Win32.Folcom.c.exe
129. Virus.Win32.Fontra.a.exe
130. Virus.Win32.Fosforo.c.exe
131. Virus.Win32.FunLove.4070.exe
132. Virus.Win32.Gaybar.exe
133. Virus.Win32.Genu.c.exe
134. Virus.Win32.Ginra.3413.exe
135. Virus.Win32.Giri.4937.c.exe
136. Virus.Win32.Gloria.2820.exe
137. Virus.Win32.Gnil.a.exe
138. Virus.Win32.Golem.a.exe
139. Virus.Win32.Gpcode.ak.exe
140. Virus.Win32.Gremo.3302.exe
141. Virus.Win32.Grum.c.exe
142. Virus.Win32.Grum.g.exe
143. Virus.Win32.Grum.l.exe
144. Virus.Win32.Gypet.6340.exe
145. Virus.Win32.Halen.2277.exe
146. Virus.Win32.Hatred.e.exe
147. Virus.Win32.Henky.1576.exe
148. Virus.Win32.Henky.3188.exe
149. Virus.Win32.Henky.720.exe
150. Virus.Win32.Heretic.1986.exe
151. Virus.Win32.Highway.b.exe
152. Virus.Win32.HIV.6680.exe
153. Virus.Win32.HLLC.Delfer.g.exe
154. Virus.Win32.HLLC.Hai.exe
155. Virus.Win32.HLLC.Nan.exe
156. Virus.Win32.HLLC.Novelce.c.exe

157. Virus.Win32.HLLC.Relaxer.exe
158. Virus.Win32.HLLC.StupRed.exe
159. Virus.Win32.HLLC.Trafix.exe
160. Virus.Win32.HLLC.Vedex.d.exe
161. Virus.Win32.HLLO.12355.exe
162. Virus.Win32.HLLO.Ant.a.exe
163. Virus.Win32.HLLO.Ant.e.exe
164. Virus.Win32.HLLO.Casbo.b.exe
165. Virus.Win32.HLLO.Enterus.c.exe
166. Virus.Win32.HLLO.Hadefix.b.exe
167. Virus.Win32.HLLO.Homer.a.exe
168. Virus.Win32.HLLO.Ivad.exe
169. Virus.Win32.HLLO.Job.22528.exe
170. Virus.Win32.HLLO.Mip.exe
171. Virus.Win32.HLLO.Ower.20480.exe
172. Virus.Win32.HLLO.Tarex.exe
173. Virus.Win32.HLLO.ZMK.b.exe
174. Virus.Win32.HLLP.Alcaul.c.exe
175. Virus.Win32.HLLP.Bizac.c.exe
176. Virus.Win32.HLLP.Cranb.a.exe
177. Virus.Win32.HLLP.Delf.a.exe
178. Virus.Win32.HLLP.Delf.e.exe
179. Virus.Win32.HLLP.Dugert.b.exe
180. Virus.Win32.HLLP.Eter.b.exe
181. Virus.Win32.HLLP.Flatei.a.exe
182. Virus.Win32.HLLP.Flatei.f.exe
183. Virus.Win32.HLLP.Ghostdog.c.exe
184. Virus.Win32.HLLP.Gotem.exe
185. Virus.Win32.HLLP.Hantaner.d.exe
186. Virus.Win32.HLLP.Lmir.a.exe
187. Virus.Win32.HLLP.Pres.exe
188. Virus.Win32.HLLP.Rosec.exe
189. Virus.Win32.HLLP.Semisoft.c.exe
190. Virus.Win32.HLLP.Semisoft.g.exe
191. Virus.Win32.HLLP.Semisoft.l.exe
192. Virus.Win32.HLLP.Shodi.b.exe
193. Virus.Win32.HLLP.Text.a.exe
194. Virus.Win32.HLLP.Tweder.exe
195. Virus.Win32.HLLP.VB.b.exe
196. Virus.Win32.HLLP.VB.j.exe
197. Virus.Win32.HLLP.Vedex.b.exe
198. Virus.Win32.HLLP.Xinfect.a.exe
199. Virus.Win32.HLLP.Xinfect.e.exe
200. Virus.Win32.HLLP.Xinfect.j.exe
201. Virus.Win32.HLLP.Yellor.b.exe
202. Virus.Win32.HLLP.Zepp.d.exe

203. Virus.Win32.HLLP.Zepp.h.exe
204. Virus.Win32.HLLW.Acoola.a.exe
205. Virus.Win32.HLLW.Alcaul.b.exe
206. Virus.Win32.HLLW.Aldax.exe
207. Virus.Win32.HLLW.AntiQFX.a.exe
208. Virus.Win32.HLLW.Archex.exe
209. Virus.Win32.HLLW.Bezilom.dr.exe
210. Virus.Win32.HLLW.Billrus.c.exe
211. Virus.Win32.HLLW.Billrus.g.exe
212. Virus.Win32.HLLW.Boomer.exe
213. Virus.Win32.HLLW.Conteo.exe
214. Virus.Win32.HLLW.Cybercer.exe
215. Virus.Win32.HLLW.Delf.d.exe
216. Virus.Win32.HLLW.Delf.k.exe
217. Virus.Win32.HLLW.Delf.p.exe
218. Virus.Win32.HLLW.Dexec.exe
219. Virus.Win32.HLLW.Emeres.exe
220. Virus.Win32.HLLW.Felic.exe
221. Virus.Win32.HLLW.Flor.exe
222. Virus.Win32.HLLW.Ghotex.a.exe
223. Virus.Win32.HLLW.Giwin.exe
224. Virus.Win32.HLLW.Karimex.exe
225. Virus.Win32.HLLW.Kimex.exe
226. Virus.Win32.HLLW.Loxar.exe
227. Virus.Win32.HLLW.Maryl.exe
228. Virus.Win32.HLLW.Mokser.exe
229. Virus.Win32.HLLW.Mouselom.exe
230. Virus.Win32.HLLW.Oblion.a.exe
231. Virus.Win32.HLLW.Osapex.a.exe
232. Virus.Win32.HLLW.pakes.exe
233. Virus.Win32.HLLW.Perdex.exe
234. Virus.Win32.HLLW.Porner.exe
235. Virus.Win32.HLLW.Randir.exe
236. Virus.Win32.HLLW.Rolog.f.exe
237. Virus.Win32.HLLW.Selfoner.exe
238. Virus.Win32.HLLW.Smilex.exe
239. Virus.Win32.HLLW.Spinex.exe
240. Virus.Win32.HLLW.Tborro.exe
241. Virus.Win32.HLLW.Timese.a.exe
242. Virus.Win32.HLLW.Trabos.a.exe
243. Virus.Win32.HLLW.Vavico.exe
244. Virus.Win32.Horope.d.exe
245. Virus.Win32.Hortiga.4800.exe
246. Virus.Win32.Htrip.b.exe
247. Virus.Win32.Idele.1839.exe
248. Virus.Win32.Idele.2160.exe

249. Virus.Win32.Idyll.1556.b.exe
250. Virus.Win32.Iframer.c.exe
251. Virus.Win32.IKX.exe
252. Virus.Win32.Infeme.exe
253. Virus.Win32.Infynca.565.exe
254. Virus.Win32.Inrar.c.exe
255. Virus.Win32.Insom.1972.b.exe
256. Virus.Win32.Instan.a.exe
257. Virus.Win32.Ipamor.a.exe
258. Virus.Win32.Ivaz.exe
259. Virus.Win32.Jeepeg.d.exe
260. Virus.Win32.Jeepeg.h.exe
261. Virus.Win32.Jeff.a.exe
262. Virus.Win32.Julikz.a.exe
263. Virus.Win32.Kate.a.exe
264. Virus.Win32.Kaze.4236.exe
265. Virus.Win32.Keisan.c.exe
266. Virus.Win32.Kenston.1895.a.exe
267. Virus.Win32.Kespy.a.exe
268. Virus.Win32.Kies.b.exe
269. Virus.Win32.Killis.a.exe
270. Virus.Win32.Klinge.exe
271. Virus.Win32.Koru.exe
272. Virus.Win32.Kriz.3689.exe
273. Virus.Win32.Kriz.4029.exe
274. Virus.Win32.Kriz.4075.exe
275. Virus.Win32.Kroter.exe
276. Virus.Win32.Kvex.a.exe
277. Virus.Win32.Lamchi.b.exe
278. Virus.Win32.Lamer.b.exe
279. Virus.Win32.Lamer.g.exe
280. Virus.Win32.Lamewin.1751.exe
281. Virus.Win32.Lamicho.b.exe
282. Virus.Win32.LAMT.exe
283. Virus.Win32.Lash.c.exe
284. Virus.Win32.Legacy.exe
285. Virus.Win32.Levi.3236.b.exe
286. Virus.Win32.Levi.3432.exe
287. Virus.Win32.Lifort.1465.exe
288. Virus.Win32.Lykov.a.exe
289. Virus.Win32.Magic.1922.exe
290. Virus.Win32.Magic.7045.a.exe
291. Virus.Win32.Magic.7045.e.exe
292. Virus.Win32.Magic.7045.i.exe
293. Virus.Win32.Mark.919.exe
294. Virus.Win32.Matrix.817.b.exe

295. Virus.Win32.Matrix.LS.1820.exe
296. Virus.Win32.Matrix.Ordy.c.exe
297. Virus.Win32.Matrix.Zelda.b.exe
298. Virus.Win32.Maya.4153.a.exe
299. Virus.Win32.Maya.4207.exe
300. Virus.Win32.Melder.exe
301. Virus.Win32.Mental.9996.exe
302. Virus.Win32.Miam.1696.exe
303. Virus.Win32.Miam.3657.exe
304. Virus.Win32.Miam.5168.exe
305. Virus.Win32.Missu.1757.exe
306. Virus.Win32.Mkar.g.exe
307. Virus.Win32.Mogul.6800.exe
308. Virus.Win32.Mohmed.4354.exe
309. Virus.Win32.Mooder.c.exe
310. Virus.Win32.Mooder.g.exe
311. Virus.Win32.Mooder.k.exe
312. Virus.Win32.MTV.4608.a.exe
313. Virus.Win32.Mudant.887.exe
314. Virus.Win32.Mystery.2560.exe
315. Virus.Win32.Nemsi.b.exe
316. Virus.Win32.Neshta.b.exe
317. Virus.Win32.NGVCK.1030.exe
318. Virus.Win32.NGVCK.4347.exe
319. Virus.Win32.Niya.a.exe
320. Virus.Win32.Nox.2290.b.exe
321. Virus.Win32.Nytra.a.exe
322. Virus.Win32.Oporto.3076.exe
323. Virus.Win32.Oroch.3982.exe
324. Virus.Win32.Paradise.2116.exe
325. Virus.Win32.Parite.b.exe
326. Virus.Win32.PayBack.1325.exe
327. Virus.Win32.Perez.b.exe
328. Virus.Win32.Perez.g.exe
329. Virus.Win32.Pioneer.a.exe
330. Virus.Win32.Plut.a.exe
331. Virus.Win32.Porex.a.exe
332. Virus.Win32.Poson.1631.exe
333. Virus.Win32.Projet.2342.exe
334. Virus.Win32.Projet.2649.exe
335. Virus.Win32.Pulkfer.b.exe
336. Virus.Win32.Qozah.3361.exe
337. Virus.Win32.Radja.a.exe
338. Virus.Win32.RainSong.3925.b.exe
339. Virus.Win32.Ramdile.exe
340. Virus.Win32.Ramm.d.exe

341. Virus.Win32.Ramm.h.exe
342. Virus.Win32.Ramm.n.exe
343. Virus.Win32.Redart.2796.exe
344. Virus.Win32.Resur.b.exe
345. Virus.Win32.Resur.f.exe
346. Virus.Win32.Revaz.exe
347. Virus.Win32.Rigel.6468.exe
348. Virus.Win32.Rotor.a.exe
349. Virus.Win32.Rufis.b.exe
350. Virus.Win32.Sabus.a.exe
351. Virus.Win32.Sality.a.exe
352. Virus.Win32.Sality.ad.exe
353. Virus.Win32.Sality.d.exe
354. Virus.Win32.Sality.j.exe
355. Virus.Win32.Sality.y.exe
356. Virus.Win32.Sandman.4096.exe
357. Virus.Win32.Sankei.3001.exe
358. Virus.Win32.Sankei.3454.exe
359. Virus.Win32.Sankei.3514.exe
360. Virus.Win32.Sankei.3621.exe
361. Virus.Win32.Savior.1696.exe
362. Virus.Win32.Savior.1832.exe
363. Virus.Win32.Segax.1161.exe
364. Virus.Win32.Seluum.a.exe
365. Virus.Win32.Seppuku.1606.exe
366. Virus.Win32.Seppuku.2764.exe
367. Virus.Win32.Seppuku.3291.exe
368. Virus.Win32.Seppuku.4831.exe
369. Virus.Win32.Seppuku.6973.exe
370. Virus.Win32.Shaitan.3392.exe
371. Virus.Win32.Shodi.e.exe
372. Virus.Win32.Shown.538.exe
373. Virus.Win32.Shown.540.b.exe
374. Virus.Win32.Siller.1364.exe
375. Virus.Win32.Silly.c.exe
376. Virus.Win32.SillyC.6006.exe
377. Virus.Win32.Sinn.1397.exe
378. Virus.Win32.Slaman.d.exe
379. Virus.Win32.Slaman.i.exe
380. Virus.Win32.Small.1338.exe
381. Virus.Win32.Small.1388.exe
382. Virus.Win32.Small.1657.exe
383. Virus.Win32.Small.2280.exe
384. Virus.Win32.Small.4096.exe
385. Virus.Win32.Small.ag.exe
386. Virus.Win32.Small.d.exe

387. Virus.Win32.Small.i.exe
388. Virus.Win32.Small.m.exe
389. Virus.Win32.Small.r.exe
390. Virus.Win32.Small.w.exe
391. Virus.Win32.Smile.a.exe
392. Virus.Win32.Smog.e.exe
393. Virus.Win32.Spit.b.exe
394. Virus.Win32.Stepar.dr.exe
395. Virus.Win32.Stepar.j.exe
396. Virus.Win32.Sugin.exe
397. Virus.Win32.Taek.1275.exe
398. Virus.Win32.Team.a.exe
399. Virus.Win32.TeddyBear.exe
400. Virus.Win32.Test.1334.exe
401. Virus.Win32.Texel.b.exe
402. Virus.Win32.Texel.f.exe
403. Virus.Win32.Texel.j.exe
404. Virus.Win32.Thorin.11932.exe
405. Virus.Win32.Thorin.c.exe
406. Virus.Win32.Tiraz.a.exe
407. Virus.Win32.Titalk.exe
408. Virus.Win32.Trats.b.exe
409. Virus.Win32.Trion.a.exe
410. Virus.Win32.Triplix.3072.b.exe
411. Virus.Win32.Tufik.b.exe
412. Virus.Win32.Tvido.b.exe
413. Virus.Win32.Ultratt.8166.a.exe
414. Virus.Win32.Undertaker.4883.b.exe
415. Virus.Win32.Usem.a.exe
416. Virus.Win32.Vangeridze.a.exe
417. Virus.Win32.VB.ac.exe
418. Virus.Win32.VB.ag.exe
419. Virus.Win32.VB.al.exe
420. Virus.Win32.VB.aq.exe
421. Virus.Win32.VB.av.exe
422. Virus.Win32.VB.az.exe
423. Virus.Win32.VB.bc.exe
424. Virus.Win32.VB.bi.exe
425. Virus.Win32.VB.bo.exe
426. Virus.Win32.VB.bs.exe
427. Virus.Win32.VB.bx.exe
428. Virus.Win32.VB.ca.exe
429. Virus.Win32.VB.cj.exe
430. Virus.Win32.VB.cn.exe
431. Virus.Win32.VB.cs.exe
432. Virus.Win32.VB.cy.exe

433. Virus.Win32.VB.de.exe
434. Virus.Win32.VB.di.exe
435. Virus.Win32.VB.dn.exe
436. Virus.Win32.VB.ds.exe
437. Virus.Win32.VB.dx.exe
438. Virus.Win32.VB.ed.exe
439. Virus.Win32.VB.ej.exe
440. Virus.Win32.VB.ep.exe
441. Virus.Win32.VB.ew.exe
442. Virus.Win32.VB.fe.exe
443. Virus.Win32.VB.fi.exe
444. Virus.Win32.VB.fo.exe
445. Virus.Win32.VB.fv.exe
446. Virus.Win32.VB.gf.exe
447. Virus.Win32.VB.gm.exe
448. Virus.Win32.VB.gu.exe
449. Virus.Win32.VB.hg.exe
450. Virus.Win32.VB.hm.exe
451. Virus.Win32.VB.hu.exe
452. Virus.Win32.VB.id.exe
453. Virus.Win32.VB.ij.exe
454. Virus.Win32.VB.ip.exe
455. Virus.Win32.VB.jg.exe
456. Virus.Win32.VB.jp.exe
457. Virus.Win32.VB.jy.exe
458. Virus.Win32.VB.ke.exe
459. Virus.Win32.VB.ko.exe
460. Virus.Win32.VB.kt.exe
461. Virus.Win32.VB.kx.exe
462. Virus.Win32.VB.lc.exe
463. Virus.Win32.VB.lj.exe
464. Virus.Win32.VB.ls.exe
465. Virus.Win32.VB.m.exe
466. Virus.Win32.VB.mg.exe
467. Virus.Win32.VB.q.exe
468. Virus.Win32.VB.u.exe
469. Virus.Win32.VbFrm.exe
470. Virus.Win32.VCell.3504.exe
471. Virus.Win32.Velost.1186.exe
472. Virus.Win32.Vibeck.7045.exe
473. Virus.Win32.Virut.ad.exe
474. Virus.Win32.Virut.ah.exe
475. Virus.Win32.Virut.ao.exe
476. Virus.Win32.Virut.as.exe
477. Virus.Win32.Virut.aw.exe
478. Virus.Win32.Virut.ba.exe

479. Virus.Win32.Virut.bf.exe
480. Virus.Win32.Virut.bl.exe
481. Virus.Win32.Virut.bq.exe
482. Virus.Win32.Virut.bv.exe
483. Virus.Win32.Virut.ca.exe
484. Virus.Win32.Virut.ce.exe
485. Virus.Win32.Virut.m.exe
486. Virus.Win32.Virut.r.exe
487. Virus.Win32.Virut.v.exe
488. Virus.Win32.Virut.z.exe
489. Virus.Win32.Vulcano.exe
490. Virus.Win32.Warmup.a.exe
491. Virus.Win32.Weird.c.exe
492. Virus.Win32.Wide.8135.a.exe
493. Virus.Win32.Wide.b.exe
494. Virus.Win32.Winemmem.a.exe
495. Virus.Win32.Wit.b.exe
496. Virus.Win32.Xorer.al.exe
497. Virus.Win32.Xorer.ax.exe
498. Virus.Win32.Xorer.bz.exe
499. Virus.Win32.Xorer.ce.exe
500. Virus.Win32.Xorer.ck.exe
501. Virus.Win32.Xorer.cp.exe
502. Virus.Win32.Xorer.cw.exe
503. Virus.Win32.Xorer.df.exe
504. Virus.Win32.Xorer.dr.exe
505. Virus.Win32.Xorer.eg.exe
506. Virus.Win32.Xorer.eo.exe
507. Virus.Win32.Xorer.es.exe
508. Virus.Win32.Xorer.ew.exe
509. Virus.Win32.Xorer.fb.exe
510. Virus.Win32.Xorer.ff.exe
511. Virus.Win32.Xorer.fk.exe
512. Virus.Win32.Xorer.m.exe
513. Virus.Win32.Xoro.4092.exe
514. Virus.Win32.Yasw.924.exe
515. Virus.Win32.Younga.2384.a.exe
516. Virus.Win32.Zaka.a.exe
517. Virus.Win32.Zaprom.2756.exe
518. Virus.Win32.Zero.a.exe
519. Virus.Win32.ZMist.ds.exe
520. Virus.Win32.Zorg.a.exe
521. Virus.Win32.ZPerm.b.exe

B.4 Benign Programs Test Set

1. 3DVision_197.45.exe
2. 3DVision_258.96.exe
3. 3DVision_296.10.exe
4. 3DVision_310.90.exe
5. 4sharedTlbr.exe
6. 7zFM.exe
7. 7zGn.exe
8. AAW2007Pro.exe
9. aaw2008_2.exe
10. AAWPro.exe
11. AAWService.exe
12. AcroRd32Info.exe
13. Adb.exe
14. aimp_2.60.525.exe
15. aimp_2.60.530.exe
16. aimp_2.61.570.exe
17. aimp_3.00.976.exe
18. AIMP2t.exe
19. alcrmv.exe
20. alcrmv9x.exe
21. ALEUpdat.exe
22. amdocl_as32.exe
23. Antivirus_Free_Edition.exe
24. ApnStub.exe
25. armsvc.exe
26. ATTv3.3.1.exe
27. AUDIOGRABBER.EXE.exe
28. AUPDRUN.EXE.exe
29. autorunsc.exe
30. avadmin.exe
31. avantvw.exe
32. avc-free_10.exe
33. avc-free_13.exe
34. avc-free_29.exe
35. avc-free_4.exe
36. avc-free_51.exe
37. avc-free_53.exe
38. avc-free_58.exe
39. avc-free_62.exe
40. avc-free_67.exe
41. avc-free_72.exe
42. avc-free_78.exe
43. avcmd.exe
44. avesvc.exe

45. avgcfgex.exe
46. AVGCTRL.EXE.exe
47. avgdiagex.exe
48. avgemcx.exe
49. AVGNT.EXE.exe
50. avgrunasx.exe
51. avgstrmx.exe
52. avscan.exe
53. AVSCHED32.EXE.exe
54. avwebg7.exe
55. AVWIN.EXE.exe
56. AxCrypt2Go.exe
57. AxDTA.exe
58. AxSrvUACHlper.exe
59. Azureus_3.0.0.8_windows.exe
60. Azureus_3.0.4.0_windows.exe
61. Azureus_3_0_windows_3.exe
62. BackItUpDCA24F76.exe
63. bcdedit.exe
64. BCompare-3.1.10.11626.exe
65. BCompare-3.1.9.11282.exe
66. BCompare-3.3.3.14128.exe
67. BCompare-3.3.5.15075.exe
68. BI.exe
69. BitTorrent_10.exe
70. BitTorrent_18.exe
71. BitTorrent_28.exe
72. BitTorrent_29.exe
73. BitTorrent_8.exe
74. bootstrap.exe
75. bspatch.exe
76. bsptb.exe
77. bundle3.exe
78. cavscan.exe
79. ccApp.exe
80. CDSpeed.exe
81. CfgWiz.exe
82. cfigm60.exe
83. cfpconfg.exe
84. chrome_launcher.exe
85. Client.exe
86. Common.exe
87. ConfigTsXP.exe
88. convert.exe
89. crashreporter.exe
90. createssoimage.exe

91. CuteWriter_4.exe
92. cwshredder_7.exe
93. daemon_mgm.exe
94. dap97_3.exe
95. date.exe
96. DeleteTemp.exe
97. dfrg.exe
98. DiagnosticsCaptureTool.exe_0407.exe
99. diff.exe
100. dlupd.exe
101. dMC-r12.2.exe
102. dMC-R13.2-Ref-Trial.exe
103. dplaunch.exe
104. dragon_updater.exe
105. drmupgds.exe
106. Dropbox.exe
107. dtshost.exe
108. EACoreCLI.exe
109. eBay_shortcuts_1016.exe
110. echo.exe
111. Ed2kLoader.exe
112. EMusicClient.exe
113. epm.exe
114. Eudora_7.1.0.7_beta.exe
115. ewidoctrl.exe
116. ExportController.exe
117. F.bin.echo.exe
118. F.bin.mysql.exe
119. F.bin.mysql_client_test_embedded.exe
120. F.bin.mysqldump.exe
121. F.bin.resolveip.exe
122. FastPictureViewer.exe
123. fd3beta_4.exe
124. fd3beta_5.exe
125. file__StartW8Menu.exe
126. flashget.exe
127. flock.exe
128. fmt.exe
129. FolderSizeSvc.exe
130. fraps.exe
131. FreemakeVideoConverter_2.0.1.2_2.exe
132. FreemakeVideoConverter_2.1.0.2_2.exe
133. FreemakeVideoConverter_2.1.4.0.exe
134. FreemakeVideoConverter_2.3.4.1_2.exe
135. FreemakeVideoConverter_3.0.0.2.exe
136. FreemakeVideoConverter_3.0.1.12.exe

137. FreemakeVideoConverter_3.0.1.13.exe
138. FreemakeVideoConverter_3.0.1.17.exe
139. FreemakeVideoConverter_3.0.1.4.exe
140. FreemakeVideoConverter_3.0.2.4_2.exe
141. FreemakeVideoConverter_3.0.2.8.exe
142. FreemakeVideoConverter_3.2.1.2.exe
143. FreemakeVideoConverter_3.2.1.7.exe
144. FreemakeVideoConverter_3.2.1.9.exe
145. FreemakeVideoConverter_4.0.0.13.exe
146. FreemakeVideoConverter_4.0.1.8_2.exe
147. FreemakeVideoConverter_4.0.2.10.exe
148. FreemakeVideoConverter_4.0.2.15_3.exe
149. FreemakeVideoConverter_4.0.2.16_2.exe
150. FWLOGCTL.EXE.exe
151. FzSFtp.exe
152. googledrivesync.exe
153. GoogleEarth4.2.196.2018.exe
154. GoogleSketchUpWEN_13.exe
155. gpsbabel.exe
156. GPU-Z.0.2.1.exe
157. GPU-Z.0.6.3.exe
158. GPU-Z.0.6.8.exe
159. GPU-Z.0.7.0.exe
160. GrLauncher.exe
161. GTB5FF.EXE.exe
162. GTBXP.EXE.exe
163. gtk-runtime.exe
164. GuardMailRu.exe
165. hao123inst-saudi-forf.exe
166. hdaudio_1.0.9.1_xp_vista_win7.exe
167. hwinfo.exe
168. ICCompressorChoose_win32.exe
169. Icon._03AA18A4B063347B01AC2E.exe
170. Icon._112D608FD02CD87FDC7735.exe
171. Icon._1585BAE38223FD8568A721.exe
172. Icon._18be6784.exe
173. Icon._2040DE605D15FF10449701.exe
174. Icon._24AB742613127F5CC6C135.exe
175. Icon._37D387FDB2BF1D95B1200F.exe
176. Icon._3B1DB7825570B7D288BB05.exe
177. Icon._5D7B49CA3A4D01C32C7C42.exe
178. Icon._702D5ACC94E574F40B7211.exe
179. Icon._717E76F86D082EE3B5B275.exe
180. Icon._756A508DD6A8A646E9E77A.exe
181. Icon._790336A8B845F6678850E5.exe
182. Icon._82E8CFA89478E80234BD70.exe

183. Icon._83E488DBE7FD86612AA106.exe
184. Icon._9C7DE206146CA6AA27B622.exe
185. Icon._A281F58C9FB96EC7E31292.exe
186. Icon._AEE1A577E347A2D10062CD.exe
187. Icon._B6788C656B2FD9CCEF44EF.exe
188. Icon._BA79C0D741BFF81B082F58.exe
189. Icon._C8F43F461AD0857274FF89.exe
190. Icon._D8ECF83B33A1F5704C1F0F.exe
191. Icon._DC60A53C789CA9E41F88BD.exe
192. Icon._DF99DDF6D2F676E469FA56.exe
193. Icon._EECCE476BD665F7A4528AF.exe
194. Icon.GalleryStartMenuSh_70DD0B26371643879C8D4168168B149E.exe
195. Icon.ico_kss.1.0.0.500.ico.exe
196. Icon.IncrediMailMenuFol_A2DA5AEC1C204AFCA02B199D5A54DAC2.exe
197. Icon.maintenance_icon.exe
198. Icon.sdraw.exe
199. Icon.swriter.exe
200. Icon.SystemFolder_msiexec.exe
201. Icon.wrdvicon.exe
202. iconworkshop_11.exe
203. iconworkshop_18.exe
204. iconworkshop_4.exe
205. iconworkshop_9.exe
206. ICS_Dv32_3.exe
207. idman607_8.exe
208. idman609.exe
209. idman612_10.exe
210. idman612_2.exe
211. idman614.exe
212. idman614_2.exe
213. idman615_10.exe
214. idman615_11.exe
215. idman615_4.exe
216. idman615_5.exe
217. IESyncClient.exe
218. iMeshV6_2.exe
219. iMeshV7_2.exe
220. iMeshV8.exe
221. InCDL166CC5F2.exe
222. infrarecorder.exe
223. inkscape.exe
224. InstanceManager.exe
225. InstMsiA.Exe.exe
226. InstMsiW.Exe.exe
227. INUNINST.exe
228. inyt.exe

229. iPlayer.exe
230. isobuster_all_lang_3.exe
231. isobuster_all_lang_4.exe
232. isobuster_all_lang_7.exe
233. iTunesHelper.exe
234. itype.exe
235. javacpl.exe
236. jre-1_5_0_07-windows-i586-p.exe
237. jre-6u3-windows-i586-p.exe
238. jre-6u5-windows-i586-p.exe
239. jre-windows-i586.exe
240. Kerio_5.exe
241. KHALMNPR.EXE.exe
242. ksomisc.exe
243. livecall.exe
244. LSDriveDetect.exe
245. lua5.1a_gui.exe
246. LuCallbackProxy.exe
247. LuComServer_3_2.EXE.exe
248. m4.exe
249. magnify.exe
250. MailWasher_Free_651.exe
251. makensis.exe
252. maxupdate.exe
253. mbsa2rix86.exe
254. MediaInfo.exe
255. MediaMonkey_3.0.2.1129.exe
256. MediaMonkey_3.0.3.1164.exe
257. MediaMonkey_3.0.3.1166.exe
258. MediaMonkey_3.0.6.1189.exe
259. MediaMonkey_3.0.7.1191.exe
260. MediaMonkey_3.1.0.1204_Debug.exe
261. MediaMonkey_3.1.0.1229_Debug.exe
262. MediaMonkey_3.2.3.1303.exe
263. MediaMonkey_4.0.2.1462.exe
264. MediaMonkey_4.0.3.1466.exe
265. MediaMonkey_4.0.5.1496.exe
266. mencoder.exe
267. mergecap.exe
268. migrator.exe
269. mirc.exe
270. mirc616.exe
271. mirc631.exe
272. mirc633.exe
273. miro-segmenter.exe
274. MirrorShim.exe

275. modern_headerbmp.exe
276. modern_nodesc.exe
277. moviethumb.exe
278. MULTIFIX.EXE.exe
279. MxUp.exe
280. ncat.exe
281. ndswapper.exe
282. nentenst_4.exe
283. Nero_KwikMedia-11.0.15300_free_2.exe
284. Nero_KwikMedia-12.5.00300_free_10.exe
285. NeroAACWrapper.exe
286. NeroCheck.exe
287. NeroDelTmp.exe
288. NeroHome65483717.exe
289. NeroHomeF0D6B0A0.exe
290. NeroMediaHome43DCD1AC.exe
291. NeroRichPreview601E7E44.exe
292. NeroSearchAdvanced3C3D1DE3.exe
293. NiReg.exe
294. NMFirstStartD9B4E50E.exe
295. NMIndexStoreSvr9AC435F3.exe
296. NMMediaServer7FBBE085.exe
297. nod32kui.exe
298. NSCSRVCE.EXE.exe
299. NSIS.exe
300. NSIS.Library.RegTool.v2.$[35].exe
301. NSIS.Library.RegTool.v3.$[102].exe
302. nt4ldr.exe
303. nvlhr.exe
304. nvStreaming.exe
305. OEMkeys.exe
306. opera_sws.exe
307. Origin.exe
308. OSE.EXE.exe
309. p98.exe
310. package_inst.exe
311. pcswpcsi.exe
312. PhotoScape.exe
313. PhotoSnapC12FC710.exe
314. PhysX_8.09.04_SystemSoftware.exe
315. PhysX_9.09.0428_SystemSoftware.exe
316. Picasa2.exe
317. PicasaCD.exe
318. PicasaPhotoViewer.exe
319. PicasaUpdater.exe
320. pifCrawl.exe

321. PMGRANT.EXE.exe
322. postgresql-9.0.4-1-windows.exe
323. postgresql-9.1.1-1-windows.exe
324. postgresql-9.2.1-1-windows.exe
325. powarc964.exe
326. PowerDVD12ML.exe
327. PowerDVD13ML.exe
328. PRFB-Chrome.exe
329. procexp.exe
330. PSANToManager.exe
331. PSPP12_Corel_TBYB_EN_IE_FR_DE_ES_IT_NL_ESD_2.exe
332. PSPP12_Corel_TBYB_EN_IE_FR_DE_ES_IT_NL_ESD_3.exe
333. qconsole.exe
334. qttask.exe
335. QuickTimeUpdateHelper.exe
336. rapimgr.exe
337. RDBVALIDATE.EXE.exe
338. RealPlayer.exe
339. RealPlayer_11.exe
340. rebasedlls.exe
341. Recode1A2D4B1C.exe
342. ReflectService.exe
343. RegCleanup.exe
344. regmerge.exe
345. Reporter.exe
346. restart_helper.exe
347. RichVideo.exe
348. rmid.exe
349. roboform.exe
350. RocketDock.exe
351. san1420.exe
352. san1572.exe
353. san1599.exe
354. san1652.exe
355. san1667_2.exe
356. san1772.exe
357. san1780.exe
358. san1828_2.exe
359. san1847.exe
360. san1874.exe
361. san1923.exe
362. san2011-1755.exe
363. SandboxieDcomLaunch.exe
364. sbase.exe
365. sbrc.exe
366. sc14.exe

367. schedul.EXE.exe
368. schk.exe
369. SDActivate.exe
370. sdbarker_tiny.exe
371. SDKCOMPONENTS_PPCRL_IDBHOSERVICE.EXE.exe
372. SecurityScan_release_small.exe
373. SetBrowser.exe
374. SetClean.exe
375. shexp.exe
376. ShrinkTo5Gui.exe
377. SKDialogsEXE.exe
378. SmeDump.exe
379. SoundTrax9C09255C.exe
380. spybotsd-1.6.1.38.exe
381. spybotsd-1.6.1.41.exe
382. spybotsd-2.0.5-beta3.exe
383. SpybotSD2.exe
384. sqlagent.exe
385. sqldiag.exe
386. sqlps.exe
387. SRSAI.exe
388. StarWindServiceAE.exe
389. stinger32_13.exe
390. stinger32_16.exe
391. stinger32_17.exe
392. stinger32_2.exe
393. stinger32_22.exe
394. stinger32_24.exe
395. stinger32_3.exe
396. stinger32_31.exe
397. stinger32_34.exe
398. stinger32_4.exe
399. stinger32_44.exe
400. stinger32_48.exe
401. stinger32_5.exe
402. stinger32_7.exe
403. stinger32_8.exe
404. stinger32_9.exe
405. sunbelt-personal-firewall_4.exe
406. SUPERAntiSpyware_104.exe
407. SUPERAntiSpyware_107.exe
408. SUPERAntiSpyware_111.exe
409. SUPERAntiSpyware_60.exe
410. SUPERAntiSpyware_61.exe
411. SUPERAntiSpyware_69.exe
412. SUPERAntiSpyware_75.exe

413. SUPERAntiSpyware_77.exe
414. SUPERAntiSpyware_86.exe
415. SUPERAntiSpyware_92.exe
416. SWDNLD.EXE.exe
417. SwHelper_1151601.exe
418. SwHelper_1161629.exe
419. SwHelper_1166636.exe
420. SwHelper_1202122.exe
421. SWINIT.EXE.exe
422. swriter.exe
423. SyncUIHandler.exe
424. TagRename361.exe
425. TagRename37.exe
426. talkback.exe
427. TcUsbRun.exe
428. teracopy222.exe
429. tinger32_14.exe
430. TopStyle50_10.exe
431. TopStyle50_15.exe
432. touchmousepractice.exe
433. TweakUI.exe
434. unit.exe
435. Unlocker.exe
436. UnlockerInject32.exe
437. uno.exe
438. UPDCLIENT.EXE.exe
439. updroots.exe
440. utorrent_251.exe
441. utorrent_269.exe
442. utorrent-1.4.2-beta-build-427.exe
443. utorrent-1.5.1-beta-build-456_2.exe
444. utorrent-1.5.1-beta-build-460.exe
445. VBoxGuestDrvInst.exe
446. vcredist2008_x86.exe
447. VirtualBox-3.0.0_BETA1-48728-Win.exe
448. VirtualBox-3.0.10-54097-Win_2.exe
449. VirtualBox-3.0.4-50677-Win.exe
450. VirtualBox-3.1.0_BETA3-55271-Win.exe
451. VirtualBox-3.1.8-61349-Win.exe
452. VirtualBox-3.2.12-68302-Win.exe
453. VirtualBox-3.2.2-62298-Win.exe
454. VirtualBox-3.2.6_BETA2-62980-Win.exe
455. VirtualBox-4.0.4-70112-Win_2.exe
456. vlc.exe
457. VNC-5.0.2-Windows_2.exe
458. VProOneTouch.exe

459. VProSvc.exe
460. Vuze_4.0.0.4_windows.exe
461. Vuze_4.0.0.4b_windows.exe
462. Vuze_4.2.0.2_windows.exe
463. Vuze_4.3.1.4_windows.exe
464. Vuze_4406a_windows.exe
465. Vuze_4510a_windows.exe
466. Vuze_4600_windows.exe
467. Vuze_4712_windows.exe
468. WAIKFiles.exe
469. WeatherBug.exe
470. wextract.exe
471. wic_x86_enu.exe
472. widgetsus_16.exe
473. windowblinds5_public.exe
474. windowblinds5_public_4.exe
475. WinSnap.exe
476. Wireless_14.3.0.6_Dv32_3.exe
477. wmaudioredist.exe
478. wmdbexport.exe
479. wmfdist11.exe
480. wmplayer.exe
481. wmpshare.exe
482. wpdshextautoplay.exe
483. wzwipe32.exe
484. Xfire.exe
485. XnView-win_16.exe
486. XnView-win_77.exe
487. ymsgr1000_542_us.exe
488. ytb_inst.exe
489. ytb3.exe
490. YzDock.exe
491. ZAPrivacyService.exe
492. zbrowser.exe
493. ZLCLIENT.EXE.exe
494. zugo-silent.exe

B.5 Malicious Programs Test Set

1. Virus.Win32.Adson.1559.exe
2. Virus.Win32.Adson.1734.exe
3. Virus.Win32.Agent.a.exe
4. Virus.Win32.Agent.am.exe
5. Virus.Win32.Agent.bf.exe
6. Virus.Win32.Agent.cf.exe
7. Virus.Win32.Agent.ck.exe

8. Virus.Win32.Agent.t.exe
9. Virus.Win32.Alcaul.f.exe
10. Virus.Win32.Alcaul.j.exe
11. Virus.Win32.Alcaul.n.exe
12. Virus.Win32.Alma.2414.exe
13. Virus.Win32.Alman.a.exe
14. Virus.Win32.Anuir.3818.exe
15. Virus.Win32.AOC.2044.exe
16. Virus.Win32.AOC.3657.exe
17. Virus.Win32.Apoc.a.exe
18. Virus.Win32.Arch.a.exe
19. Virus.Win32.Arrow.c.exe
20. Virus.Win32.Assill.a.exe
21. Virus.Win32.Auryn.1157.exe
22. Virus.Win32.AutoRun.akv.exe
23. Virus.Win32.AutoWorm.3072.exe
24. Virus.Win32.Awfull.3571.exe
25. Virus.Win32.Bakaver.a.exe
26. Virus.Win32.Banaw.2157.exe
27. Virus.Win32.Bayan.a.exe
28. Virus.Win32.Belial.2537.exe
29. Virus.Win32.Belial.c.exe
30. Virus.Win32.Belod.b.exe
31. Virus.Win32.Benny.3219.b.exe
32. Virus.Win32.BHO.c.exe
33. Virus.Win32.Bika.1906.exe
34. Virus.Win32.Blakan.2020.exe
35. Virus.Win32.Blateroz.exe
36. Virus.Win32.Bobep.exe
37. Virus.Win32.Bolzano.2676.exe
38. Virus.Win32.Bolzano.3148.exe
39. Virus.Win32.Bolzano.3904.exe
40. Virus.Win32.Bolzano.4096.d.exe
41. Virus.Win32.Bolzano.5396.b.exe
42. Virus.Win32.Brof.b.exe
43. Virus.Win32.Bube.c.exe
44. Virus.Win32.Bube.h.exe
45. Virus.Win32.Bube.m.exe
46. Virus.Win32.Bytesv.1481.b.exe
47. Virus.Win32.Cabanas.c.exe
48. Virus.Win32.CabInfector.exe
49. Virus.Win32.Cargo.b.exe
50. Virus.Win32.Ceel.c.exe
51. Virus.Win32.Champ.5477.exe
52. Virus.Win32.Champ.5707.a.exe
53. Virus.Win32.Champ.7001.exe

54. Virus.Win32.Chimera.a.exe
55. Virus.Win32.Chiton.e.exe
56. Virus.Win32.Chiton.k.exe
57. Virus.Win32.Chiton.p.exe
58. Virus.Win32.Chiton.u.exe
59. Virus.Win32.Chuzy.b.exe
60. Virus.Win32.Cmay.1222.exe
61. Virus.Win32.Cream.a.exe
62. Virus.Win32.Crunk.b.exe
63. Virus.Win32.Crypto.exe
64. Virus.Win32.CTZA.d.exe
65. Virus.Win32.Damm.1624.exe
66. Virus.Win32.Damm.1796.exe
67. Virus.Win32.Daum.a.exe
68. Virus.Win32.Delf.aa.exe
69. Virus.Win32.Delf.af.exe
70. Virus.Win32.Delf.ak.exe
71. Virus.Win32.Delf.ao.exe
72. Virus.Win32.Delf.as.exe
73. Virus.Win32.Delf.bc.exe
74. Virus.Win32.Delf.bl.exe
75. Virus.Win32.Delf.bs.exe
76. Virus.Win32.Delf.ce.exe
77. Virus.Win32.Delf.ct.exe
78. Virus.Win32.Delf.de.exe
79. Virus.Win32.Delf.k.exe
80. Virus.Win32.Delf.p.exe
81. Virus.Win32.Delf.u.exe
82. Virus.Win32.Delikon.exe
83. Virus.Win32.Dicomp.8192.a.exe
84. Virus.Win32.Dion.a.exe
85. Virus.Win32.Doser.4183.exe
86. Virus.Win32.Doser.4535.exe
87. Virus.Win32.Doser.4542.exe
88. Virus.Win32.Downloader.ac.exe
89. Virus.Win32.Downloader.ah.exe
90. Virus.Win32.Downloader.am.exe
91. Virus.Win32.Downloader.aq.exe
92. Virus.Win32.Downloader.av.exe
93. Virus.Win32.Downloader.be.exe
94. Virus.Win32.Downloader.bj.exe
95. Virus.Win32.Downloader.e.exe
96. Virus.Win32.Downloader.l.exe
97. Virus.Win32.Downloader.p.exe
98. Virus.Win32.Downloader.w.exe
99. Virus.Win32.Dream.4916.exe

100. Virus.Win32.Drol.5337.a.exe
101. Virus.Win32.Drowor.b.exe
102. Virus.Win32.DunDun.1396.exe
103. Virus.Win32.Dzan.c.exe
104. Virus.Win32.Eggroll.a.exe
105. Virus.Win32.Egolet.d.exe
106. Virus.Win32.Elkern.c.exe
107. Virus.Win32.Emotion.a.exe
108. Virus.Win32.Emotion.gen.exe
109. Virus.Win32.Enerlam.c.exe
110. Virus.Win32.Enumiacs.8192.b.exe
111. Virus.Win32.Eva.a.exe
112. Virus.Win32.Evar.3587.exe
113. Virus.Win32.Evul.8192.a.exe
114. Virus.Win32.Evul.8192.e.exe
115. Virus.Win32.Evyl.a.exe
116. Virus.Win32.Evyl.h.exe
117. Virus.Win32.Expiro.d.exe
118. Virus.Win32.Expiro.h.exe
119. Virus.Win32.Expiro.m.exe
120. Virus.Win32.Falkonder.exe
121. Virus.Win32.Fighter.b.exe
122. Virus.Win32.FlyStudio.b.exe
123. Virus.Win32.Folcom.d.exe
124. Virus.Win32.Fontra.c.exe
125. Virus.Win32.Fosforo.d.exe
126. Virus.Win32.FunLove.dam.exe
127. Virus.Win32.gen.exe
128. Virus.Win32.Genu.d.exe
129. Virus.Win32.Ginra.3570.exe
130. Virus.Win32.Giri.4919.exe
131. Virus.Win32.Giri.4970.exe
132. Virus.Win32.Gloria.2928.exe
133. Virus.Win32.Gobi.a.exe
134. Virus.Win32.Goli.a.exe
135. Virus.Win32.Grum.d.exe
136. Virus.Win32.Grum.h.exe
137. Virus.Win32.Grum.m.exe
138. Virus.Win32.Harrier.exe
139. Virus.Win32.Henky.1632.exe
140. Virus.Win32.Henky.504.exe
141. Virus.Win32.Henky.772.a.exe
142. Virus.Win32.Hezhi.exe
143. Virus.Win32.HIV.6340.exe
144. Virus.Win32.HLL.Fugo.exe
145. Virus.Win32.HLLC.Delfer.a.exe

146. Virus.Win32.HLLC.Ext.exe
147. Virus.Win32.HLLC.Hiderec.exe
148. Virus.Win32.HLLC.Nosyst.exe
149. Virus.Win32.HLLC.Persian.exe
150. Virus.Win32.HLLC.Roex.exe
151. Virus.Win32.HLLC.Sulpex.a.exe
152. Virus.Win32.HLLC.Vbinfer.a.exe
153. Virus.Win32.HLLC.Vedex.f.exe
154. Virus.Win32.HLLO.28471.exe
155. Virus.Win32.HLLO.Ant.b.exe
156. Virus.Win32.HLLO.Antim.exe
157. Virus.Win32.HLLO.Cewalk.exe
158. Virus.Win32.HLLO.Fad.exe
159. Virus.Win32.HLLO.Hadefix.d.exe
160. Virus.Win32.HLLO.Homer.b.exe
161. Virus.Win32.HLLO.Jetto.a.exe
162. Virus.Win32.HLLO.Pascol.exe
163. Virus.Win32.HLLO.Thiz.17408.exe
164. Virus.Win32.HLLO.ZMK.c.exe
165. Virus.Win32.HLLP.Alcaul.d.exe
166. Virus.Win32.HLLP.BadBy.exe
167. Virus.Win32.HLLP.Bizac.d.exe
168. Virus.Win32.HLLP.Cranb.b.exe
169. Virus.Win32.HLLP.Delf.b.exe
170. Virus.Win32.HLLP.Delvi.exe
171. Virus.Win32.HLLP.Emesix.exe
172. Virus.Win32.HLLP.Eter.c.exe
173. Virus.Win32.HLLP.Flatei.c.exe
174. Virus.Win32.HLLP.Gezad.exe
175. Virus.Win32.HLLP.Hantaner.a.exe
176. Virus.Win32.HLLP.Hantaner.e.exe
177. Virus.Win32.HLLP.Kiro.exe
178. Virus.Win32.HLLP.Metrion.a.exe
179. Virus.Win32.HLLP.MTV.exe
180. Virus.Win32.HLLP.Remcom.exe
181. Virus.Win32.HLLP.Savno.exe
182. Virus.Win32.HLLP.Semisoft.d.exe
183. Virus.Win32.HLLP.Semisoft.h.exe
184. Virus.Win32.HLLP.Semisoft.m.exe
185. Virus.Win32.HLLP.Shodi.c.exe
186. Virus.Win32.HLLP.Stagol.a.exe
187. Virus.Win32.HLLP.Tamin.exe
188. Virus.Win32.HLLP.Text.b.exe
189. Virus.Win32.HLLP.Unzi.exe
190. Virus.Win32.HLLP.VB.c.exe
191. Virus.Win32.HLLP.VB.k.exe

192. Virus.Win32.HLLP.Vedex.c.exe
193. Virus.Win32.HLLP.Xinfect.b.exe
194. Virus.Win32.HLLP.Xinfect.f.exe
195. Virus.Win32.HLLP.Zepp.a.exe
196. Virus.Win32.HLLP.Zepp.e.exe
197. Virus.Win32.HLLP.Zepp.j.exe
198. Virus.Win32.HLLW.Acoola.b.exe
199. Virus.Win32.HLLW.Alcaul.c.exe
200. Virus.Win32.HLLW.Amivida.exe
201. Virus.Win32.HLLW.AntiQFX.b.exe
202. Virus.Win32.HLLW.Arnger.exe
203. Virus.Win32.HLLW.Axatak.exe
204. Virus.Win32.HLLW.Bifox.exe
205. Virus.Win32.HLLW.Billrus.d.exe
206. Virus.Win32.HLLW.Billrus.h.exe
207. Virus.Win32.HLLW.Carpeta.b.exe
208. Virus.Win32.HLLW.Crumpet.exe
209. Virus.Win32.HLLW.Dabrat.exe
210. Virus.Win32.HLLW.Delf.e.exe
211. Virus.Win32.HLLW.Delf.l.exe
212. Virus.Win32.HLLW.Delf.q.exe
213. Virus.Win32.HLLW.Doxin.exe
214. Virus.Win32.HLLW.Estrella.73728.exe
215. Virus.Win32.HLLW.Filk.exe
216. Virus.Win32.HLLW.Foxma.exe
217. Virus.Win32.HLLW.Ghotex.b.exe
218. Virus.Win32.HLLW.Habrack.exe
219. Virus.Win32.HLLW.Juegos.exe
220. Virus.Win32.HLLW.Kaz.28672.exe
221. Virus.Win32.HLLW.Labirint.exe
222. Virus.Win32.HLLW.Maka.exe
223. Virus.Win32.HLLW.Mintop.a.exe
224. Virus.Win32.HLLW.Mona.exe
225. Virus.Win32.HLLW.Mousemun.exe
226. Virus.Win32.HLLW.Myset.a.exe
227. Virus.Win32.HLLW.Oblion.b.exe
228. Virus.Win32.HLLW.Osapex.b.exe
229. Virus.Win32.HLLW.Picturex.exe
230. Virus.Win32.HLLW.Remat.exe
231. Virus.Win32.HLLW.Sakao.exe
232. Virus.Win32.HLLW.Setex.exe
233. Virus.Win32.HLLW.SoftSix.a.exe
234. Virus.Win32.HLLW.Starfil.exe
235. Virus.Win32.HLLW.Tefuss.b.exe
236. Virus.Win32.HLLW.Timese.b.exe
237. Virus.Win32.HLLW.Timese.f.exe

238. Virus.Win32.HLLW.Trabos.b.exe
239. Virus.Win32.HLLW.VB.a.exe
240. Virus.Win32.HLLW.Viguito.exe
241. Virus.Win32.HLLW.Yanen.exe
242. Virus.Win32.Horope.e.exe
243. Virus.Win32.Hortiga.4805.exe
244. Virus.Win32.Htrip.c.exe
245. Virus.Win32.Idele.2060.exe
246. Virus.Win32.Idele.2219.exe
247. Virus.Win32.Idyll.1556.c.exe
248. Virus.Win32.Iframer.d.exe
249. Virus.Win32.Infinite.1661.exe
250. Virus.Win32.Initx.exe
251. Virus.Win32.Inrar.d.exe
252. Virus.Win32.Insom.1972.c.exe
253. Virus.Win32.Inta.1676.exe
254. Virus.Win32.Intar.1920.exe
255. Virus.Win32.Ipamor.b.exe
256. Virus.Win32.Jater.exe
257. Virus.Win32.Jeepeg.e.exe
258. Virus.Win32.Jeepeg.j.exe
259. Virus.Win32.Junkcomp.exe
260. Virus.Win32.Kanban.a.exe
261. Virus.Win32.Katomik.a.exe
262. Virus.Win32.Keisan.d.exe
263. Virus.Win32.Kenston.1895.b.exe
264. Virus.Win32.Kespy.b.exe
265. Virus.Win32.Kies.c.exe
266. Virus.Win32.Kiltex.exe
267. Virus.Win32.KME.exe
268. Virus.Win32.Krepper.30760.exe
269. Virus.Win32.Kriz.4037.exe
270. Virus.Win32.Kriz.4099.exe
271. Virus.Win32.Kuto.1468.exe
272. Virus.Win32.Lalexa.b.exe
273. Virus.Win32.Lamchi.c.exe
274. Virus.Win32.Lamer.c.exe
275. Virus.Win32.Lamer.h.exe
276. Virus.Win32.Lamewin.1813.exe
277. Virus.Win32.Lamicho.d.exe
278. Virus.Win32.Lamzan.exe
279. Virus.Win32.Lash.d.exe
280. Virus.Win32.Levi.3236.exe
281. Virus.Win32.Libertine.b.exe
282. Virus.Win32.Limper.exe
283. Virus.Win32.Lykov.b.exe

284. Virus.Win32.Magic.3038.exe
285. Virus.Win32.Magic.7045.b.exe
286. Virus.Win32.Magic.7045.f.exe
287. Virus.Win32.Magic.7045.j.exe
288. Virus.Win32.Mark.926.exe
289. Virus.Win32.Matrix.844.exe
290. Virus.Win32.Matrix.LS.1885.exe
291. Virus.Win32.Matrix.Ordy.d.exe
292. Virus.Win32.Matrix.Zelda.c.exe
293. Virus.Win32.Maya.4108.exe
294. Virus.Win32.Maya.4153.b.exe
295. Virus.Win32.Maya.4254.exe
296. Virus.Win32.Mental.10000.exe
297. Virus.Win32.Mental.exe
298. Virus.Win32.Miam.1699.exe
299. Virus.Win32.Miam.4716.exe
300. Virus.Win32.Minit.a.exe
301. Virus.Win32.Mkar.c.exe
302. Virus.Win32.Mocar.a.exe
303. Virus.Win32.Mogul.6806.exe
304. Virus.Win32.Mohmed.4607.exe
305. Virus.Win32.Mooder.d.exe
306. Virus.Win32.Mooder.h.exe
307. Virus.Win32.Mooder.l.exe
308. Virus.Win32.MTV.4608.b.exe
309. Virus.Win32.Munga.a.exe
310. Virus.Win32.Nemsi.exe
311. Virus.Win32.Netlip.exe
312. Virus.Win32.NGVCK.1095.exe
313. Virus.Win32.NGVCK.gen.exe
314. Virus.Win32.Nopti.exe
315. Virus.Win32.Nox.2290.exe
316. Virus.Win32.Numrok.1478.exe
317. Virus.Win32.Opdoc.1122.exe
318. Virus.Win32.Oroch.5420.exe
319. Virus.Win32.Paradise.2168.exe
320. Virus.Win32.Parite.c.exe
321. Virus.Win32.Peana.exe
322. Virus.Win32.Perez.c.exe
323. Virus.Win32.Perrun.a.exe
324. Virus.Win32.Pesin.d.exe
325. Virus.Win32.Pioneer.b.exe
326. Virus.Win32.Plutor.a.exe
327. Virus.Win32.Porex.b.exe
328. Virus.Win32.Projet.2404.a.exe
329. Virus.Win32.Qozah.1386.exe

330. Virus.Win32.Qozah.3365.exe
331. Virus.Win32.RainSong.3891.exe
332. Virus.Win32.RainSong.3956.exe
333. Virus.Win32.Ramm.a.exe
334. Virus.Win32.Ramm.e.exe
335. Virus.Win32.Ramm.i.exe
336. Virus.Win32.Ravs.a.exe
337. Virus.Win32.Redemption.b.exe
338. Virus.Win32.Resur.c.exe
339. Virus.Win32.Resur.g.exe
340. Virus.Win32.Rever.exe
341. Virus.Win32.Romario.a.exe
342. Virus.Win32.Ruff.4859.exe
343. Virus.Win32.Rufoll.1432.exe
344. Virus.Win32.Rutern.5244.exe
345. Virus.Win32.Sadon.900.exe
346. Virus.Win32.Sality.aa.exe
347. Virus.Win32.Sality.ae.exe
348. Virus.Win32.Sality.f.exe
349. Virus.Win32.Sality.o.exe
350. Virus.Win32.Sality.u.exe
351. Virus.Win32.Sality.z.exe
352. Virus.Win32.Sankei.1062.exe
353. Virus.Win32.Sankei.1493.exe
354. Virus.Win32.Sankei.3077.exe
355. Virus.Win32.Sankei.3464.exe
356. Virus.Win32.Sankei.358.exe
357. Virus.Win32.Sankei.4085.exe
358. Virus.Win32.Satir.994.exe
359. Virus.Win32.Savior.1740.exe
360. Virus.Win32.Savior.1904.exe
361. Virus.Win32.Segax.1136.exe
362. Virus.Win32.Selfish.a.exe
363. Virus.Win32.Selfish.g.exe
364. Virus.Win32.Sentinel.a.exe
365. Virus.Win32.Seppuku.1608.exe
366. Virus.Win32.Seppuku.28114.exe
367. Virus.Win32.Seppuku.3426.exe
368. Virus.Win32.Seppuku.5019.exe
369. Virus.Win32.Sexy.a.exe
370. Virus.Win32.Shaitan.3482.exe
371. Virus.Win32.Shodi.f.exe
372. Virus.Win32.Shown.539.a.exe
373. Virus.Win32.Silcer.poly.exe
374. Virus.Win32.Siller.1455.exe
375. Virus.Win32.Silly.d.exe

376. Virus.Win32.Simer.a.exe
377. Virus.Win32.Skorzen.exe
378. Virus.Win32.Slaman.f.exe
379. Virus.Win32.Small.1365.b.exe
380. Virus.Win32.Small.139.exe
381. Virus.Win32.Small.1468.exe
382. Virus.Win32.Small.1689.exe
383. Virus.Win32.Small.2560.exe
384. Virus.Win32.Small.a.exe
385. Virus.Win32.Small.ah.exe
386. Virus.Win32.Small.am.exe
387. Virus.Win32.Small.e.exe
388. Virus.Win32.Small.n.exe
389. Virus.Win32.Small.x.exe
390. Virus.Win32.Smog.a.exe
391. Virus.Win32.Spit.c.exe
392. Virus.Win32.Staro.1538.exe
393. Virus.Win32.Stepar.e.exe
394. Virus.Win32.Stepar.m.exe
395. Virus.Win32.Studen.c.exe
396. Virus.Win32.Suns.3912.exe
397. Virus.Win32.Tank.a.exe
398. Virus.Win32.Team.b.exe
399. Virus.Win32.Tenga.a.exe
400. Virus.Win32.Texel.c.exe
401. Virus.Win32.Texel.k.exe
402. Virus.Win32.Thorin.d.exe
403. Virus.Win32.Tinit.a.exe
404. Virus.Win32.Tirtas.5216.exe
405. Virus.Win32.Toex.a.exe
406. Virus.Win32.Toto.7272.exe
407. Virus.Win32.Trats.c.exe
408. Virus.Win32.Trion.b.exe
409. Virus.Win32.Tufik.c.exe
410. Virus.Win32.Tyhos.a.exe
411. Virus.Win32.Ultratt.8166.b.exe
412. Virus.Win32.Undertaker.4887.exe
413. Virus.Win32.Usem.b.exe
414. Virus.Win32.VB.ad.exe
415. Virus.Win32.VB.ah.exe
416. Virus.Win32.VB.am.exe
417. Virus.Win32.VB.ar.exe
418. Virus.Win32.VB.aw.exe
419. Virus.Win32.VB.b.exe
420. Virus.Win32.VB.bd.exe
421. Virus.Win32.VB.bl.exe

422. Virus.Win32.VB.bp.exe
423. Virus.Win32.VB.by.exe
424. Virus.Win32.VB.cb.exe
425. Virus.Win32.VB.cg.exe
426. Virus.Win32.VB.ck.exe
427. Virus.Win32.VB.cp.exe
428. Virus.Win32.VB.ct.exe
429. Virus.Win32.VB.cz.exe
430. Virus.Win32.VB.dj.exe
431. Virus.Win32.VB.do.exe
432. Virus.Win32.VB.dt.exe
433. Virus.Win32.VB.dy.exe
434. Virus.Win32.VB.ek.exe
435. Virus.Win32.VB.et.exe
436. Virus.Win32.VB.ey.exe
437. Virus.Win32.VB.ff.exe
438. Virus.Win32.VB.fj.exe
439. Virus.Win32.VB.fq.exe
440. Virus.Win32.VB.fx.exe
441. Virus.Win32.VB.hh.exe
442. Virus.Win32.VB.hz.exe
443. Virus.Win32.VB.ie.exe
444. Virus.Win32.VB.ik.exe
445. Virus.Win32.VB.ir.exe
446. Virus.Win32.VB.ja.exe
447. Virus.Win32.VB.jk.exe
448. Virus.Win32.VB.jq.exe
449. Virus.Win32.VB.k.exe
450. Virus.Win32.VB.kf.exe
451. Virus.Win32.VB.kj.exe
452. Virus.Win32.VB.kp.exe
453. Virus.Win32.VB.ku.exe
454. Virus.Win32.VB.ky.exe
455. Virus.Win32.VB.ld.exe
456. Virus.Win32.VB.lk.exe
457. Virus.Win32.VB.lw.exe
458. Virus.Win32.VB.ma.exe
459. Virus.Win32.VB.mi.exe
460. Virus.Win32.VB.r.exe
461. Virus.Win32.VB.w.exe
462. Virus.Win32.VCell.3041.exe
463. Virus.Win32.VChain.exe
464. Virus.Win32.Velost.1233.exe
465. Virus.Win32.Virut.aa.exe
466. Virus.Win32.Virut.ae.exe
467. Virus.Win32.Virut.ai.exe

468. Virus.Win32.Virut.ap.exe
469. Virus.Win32.Virut.at.exe
470. Virus.Win32.Virut.ax.exe
471. Virus.Win32.Virut.bc.exe
472. Virus.Win32.Virut.bg.exe
473. Virus.Win32.Virut.bn.exe
474. Virus.Win32.Virut.br.exe
475. Virus.Win32.Virut.bw.exe
476. Virus.Win32.Virut.cb.exe
477. Virus.Win32.Virut.cf.exe
478. Virus.Win32.Virut.n.exe
479. Virus.Win32.Virut.s.exe
480. Virus.Win32.Virut.w.exe
481. Virus.Win32.Viset.a.exe
482. Virus.Win32.Vulgar.a.exe
483. Virus.Win32.Wanhope.1357.exe
484. Virus.Win32.Warray.exe
485. Virus.Win32.Weird.d.exe
486. Virus.Win32.Wide.8135.b.exe
487. Virus.Win32.Wide.c.exe
488. Virus.Win32.Wolf.b.exe
489. Virus.Win32.Wuke.c.exe
490. Virus.Win32.Xiao.e.exe
491. Virus.Win32.Xorer.ac.exe
492. Virus.Win32.Xorer.bq.exe
493. Virus.Win32.Xorer.cf.exe
494. Virus.Win32.Xorer.cl.exe
495. Virus.Win32.Xorer.cx.exe
496. Virus.Win32.Xorer.dc.exe
497. Virus.Win32.Xorer.dj.exe
498. Virus.Win32.Xorer.ds.exe
499. Virus.Win32.Xorer.dy.exe
500. Virus.Win32.Xorer.ed.exe
501. Virus.Win32.Xorer.eh.exe
502. Virus.Win32.Xorer.el.exe
503. Virus.Win32.Xorer.ep.exe
504. Virus.Win32.Xorer.et.exe
505. Virus.Win32.Xorer.ex.exe
506. Virus.Win32.Xorer.fc.exe
507. Virus.Win32.Xorer.fg.exe
508. Virus.Win32.Xorer.h.exe
509. Virus.Win32.Xorer.s.exe
510. Virus.Win32.Yerg.9412.exe
511. Virus.Win32.Zaka.b.exe
512. Virus.Win32.Zawex.1840.exe
513. Virus.Win32.Zero.b.exe
514. Virus.Win32.ZloyFly.a.exe
515. Virus.Win32.ZMist.exe
516. Virus.Win32.Zori.a.exe
517. Virus.Win32.ZPerm.b2.exe

B.6 Additional Benign Programs List (in the Updated DB)

1. 3DVision_190.38.exe
2. 3DVision_195.62.exe
3. 3DVision_275.33.exe
4. 7z.exe
5. aaw2007_10.exe
6. aaw2007_16.exe
7. acdsee_7.exe
8. acdsee-12-0-344-win-en.exe
9. acdsee-14-0-110-win-en.exe
10. ACID.exe
11. adobearm.exe
12. adobearmhelper.exe
13. AIMLangen-US.exe
14. aimp_2.51.320.exe
15. aimp_2.60.486_beta_2.exe
16. aimp_2.61.583.exe
17. aimp_3.00.985.exe
18. AIMP2r.exe
19. ALUSchedulerSvc.exe
20. appremover_cli.exe
21. AptanaStudio3.exe
22. avcenter.exe
23. avc-free_22.exe
24. avc-free_84.exe
25. avconfig.exe
26. avguard.exe
27. avidemux_cli.exe
28. avmailc.exe
29. Azureus_3.0.1.4_windows.exe
30. Azureus_3.0.5.2b_windows.exe
31. BarBroker.exe
32. BCompare-3.2.1.12831.exe
33. beta_5.exe
34. BitTorrent_12.exe
35. BitTorrent_15.exe
36. blender.exe
37. brs.exe
38. bsetutil.exe
39. capinfos.exe
40. cfplogvw.exe
41. chkupd.exe
42. CLHNServiceForPowerDVD12.exe
43. configimport.exe
44. CPES_CLEAN.EXE.exe

45. CuteWriter_3.exe
46. daemon347_4.exe
47. dap97_2.exe
48. DefaultSettings.exe
49. DELUS.EXE.exe
50. DexControl.exe
51. DIAGNOSTICSCAPTURETOOL.EXE.exe
52. dMC-r12.3.exe
53. dMC-r12.4.exe
54. dns_sd.exe
55. DVDNavExt.exe
56. DX9.exe
57. DXEnumD7927B84.exe
58. ecls.exe
59. epm_6.exe
60. Eudora_7.0.0.15_beta_2.exe
61. Eudora_7.1.0.4_beta_2.exe
62. Eudora_7.1.0.9.exe
63. Evernote_3.1.0.1225_3.exe
64. Evernote_3.5.0.1258.exe
65. Evernote_3.5.5.2567_2.exe
66. F.bin.my_print_defaults.exe
67. F.bin.myisam_ftdump.exe
68. F.bin.myisamchk.exe
69. F.bin.mysql_tzinfo_to_sql.exe
70. F.bin.mysqlbinlog.exe
71. F.bin.mysqlslap.exe
72. ffmpeg2theora.exe
73. FGResDetector.exe
74. find.exe
75. FreemakeVideoConverter_2.1.1.1.exe
76. FreemakeVideoConverter_2.3.0.1.exe
77. FreemakeVideoConverter_2.3.1.0_2.exe
78. FreemakeVideoConverter_3.0.1.11.exe
79. FreemakeVideoConverter_3.0.2.10.exe
80. FreemakeVideoConverter_3.0.2.14.exe
81. FreemakeVideoConverter_3.2.1.6_2.exe
82. FreemakeVideoConverter_4.0.1.0.exe
83. FreemakeVideoConverter_4.0.1.1_2.exe
84. FreemakeVideoConverter_4.0.1.2_2.exe
85. FreemakeVideoConverter_4.0.2.5.exe
86. FreemakeVideoConverter_4.0.2.8.exe
87. FreemakeVideoConverter_4.0.3.1_2.exe
88. fwplayer.exe
89. gawk.exe
90. gdk-pixbuf-query-loaders.exe

91. GFExperience.exe
92. GoogleEarthWin_30.exe
93. GoogleEarthWin_EARD.exe
94. GPU-Z.0.4.5.exe
95. GPU-Z.0.4.6_2.exe
96. GPU-Z.0.5.3.exe
97. GPU-Z.0.5.6_2.exe
98. GPU-Z.0.6.0.exe
99. gtk-query-immodules-2.0.exe
100. guardgui.exe
101. GUP.exe
102. handlecmsg.exe
103. hdaudio_1.00.00.59_xp_vista_win7.exe
104. hjsplit.exe
105. HostFileEditor.exe
106. hwinfo4E43DC63.exe
107. Icon._060B8769AED00B7FE8F8AC.exe
108. Icon._12db153c.exe
109. Icon._275548FC5187708975B162.exe
110. Icon._379D30BBDA42A4EAAA996D.exe
111. Icon._5DEC29AE26BBD9C9863F82.exe
112. Icon._80DA8F2561DA44290F5B83.exe
113. Icon._8581471E17CEB84267F05B.exe
114. Icon._8862C376C51C56CA6D0BBA.exe
115. Icon._A5739470FE75C8F21815D5.exe
116. Icon._C9C70F774117D0AF316643.exe
117. Icon._EA5D8D5AF3DCEFAE240693.exe
118. Icon._F2473FCCAE2EA281EE98D5.exe
119. Icon.eula.exe
120. Icon.ImgToVHD.exe
121. Icon.MmDefaultProductIcon.1.0.0.468.ico.exe
122. iconworkshop_16.exe
123. iconworkshop_8.exe
124. ICS_Dv32.exe
125. idman611.exe
126. idman615_7.exe
127. idman615b15.exe
128. idman617b8.exe
129. ImgBurnPreview.exe
130. InCDsrv58BA8959.exe
131. isobuster_all_lang_5.exe
132. jbig2dec.exe
133. jp2launcher.exe
134. kinit.exe
135. latency.exe
136. Launcher_x64.exe

137. LavasoftGCHelper.exe
138. logname.exe
139. LuConfig.exe
140. magnet.exe
141. MailWasherFree_6.3.exe
142. MediaMonkey_3.0.4.1185.exe
143. MediaMonkey_3.1.0.1242_Debug.exe
144. MediaMonkey_3.1.0.1256.exe
145. MediaMonkey_3.2.2.1300.exe
146. mirc632.exe
147. mirc635.exe
148. mirc71.exe
149. mirc715.exe
150. Miro_Downloader.exe
151. MSOHTMED.EXE.exe
152. MSOICONS.EXE.exe
153. MxDock.exe
154. NBR.exe
155. ndntenst.exe
156. NMIndexStoreSvr9321C9AF.exe
157. NPSWF32_FlashUtil.exe
158. Offercast2802_SPC2_.exe
159. Omigrate.exe
160. PF-Chrome-W78.exe
161. PicasaUpdate.exe
162. powarc964_2.exe
163. PSANCU.exe
164. pyw.exe
165. qs.exe
166. rdcq.exe
167. RealPlayer_13.exe
168. RealPlayer_3.exe
169. RealPlayer_5.exe
170. reporter.exe
171. RunCmdFile.exe
172. SafariSyncClient.exe
173. safeguard.exe
174. SAM.exe
175. san1122.exe
176. san1715.exe
177. san1866.exe
178. san1935.exe
179. san2011-1760.exe
180. san2011-1764.exe
181. SCSIinst.exe
182. SetCDfmt.exe

183. SevenDex.exe
184. sfbeta430-9.exe
185. shlibsign.exe
186. simpress.exe
187. snagit_2.exe
188. snapshot.exe
189. SNDInst.exe
190. sobuster_all_lang_23.exe
191. Speccy.exe
192. spybotsd15he-beta1.exe
193. spybotsd160-rc2.exe
194. spybotsd-2.0.3-beta1.exe
195. SpywareTerminator_8.exe
196. Stinger_4.exe
197. stinger32_42.exe
198. Sunbelt-Personal-Firewall_2.exe
199. SUPERAntiSpyware_103.exe
200. SUPERAntiSpyware_41.exe
201. SUPERAntiSpyware_80.exe
202. SUPERAntiSpyware_91.exe
203. SUPERAntiSpyware_93.exe
204. SwHelper_1152602.exe
205. SwHelper_1156606.exe
206. SwHelper_1159620.exe
207. SwHelper_1200112.exe
208. TagRename36.exe
209. TaskSwitchXP.exe
210. ThreatWork.exe
211. thunderbird.exe
212. timidity.exe
213. ToDoList.exe
214. TopStyle50_4.exe
215. udefrag.exe
216. UninstWaDetect.exe
217. UpdateInst.exe
218. updrgui.exe
219. URLPROXY.EXE.exe
220. UsrPrmpt.exe
221. utorrent_275.exe
222. utorrent-1.5.1-beta-build-464.exe
223. video-editor_full846_2.exe
224. virtualdj_home.exe
225. vlc-cache-gen.exe
226. VMware-player-3.1.0-261024.exe
227. VoiceFrm.exe
228. Vuze_3.1.0.0_windows.exe

229. Vuze_3.1.1.0_windows.exe
230. Vuze_4.1.0.4_windows.exe
231. w9xpopen.exe
232. wia.exe
233. windowblinds5_public_6.exe
234. WindowBlinds7_public.exe
235. winpcap-nmap-4.12.exe
236. wmad.exe
237. wmlaunch.exe
238. WPMMAPI.EXE.exe
239. XCrashReport.exe
240. XnViewMediaDetector.exe
241. xpidl.exe
242. xpt_dump.exe
243. YahooWidgets_3.0.exe
244. ymsgr750_647_us.exe
245. zatray.exe
246. ZATUTOR.EXE.exe
247. zipper.exe
248. zonealarm_base.exe
249. zsh.exe
250. ZSMessage.exe

B.7 Additional Malicious Programs List (in the Updated DB)

1. Virus.Win32.Adson.1651.exe
2. Virus.Win32.Agent.ab.exe
3. Virus.Win32.Agent.bi.exe
4. Virus.Win32.Agent.cg.exe
5. Virus.Win32.Alcaul.g.exe
6. Virus.Win32.Alcaul.o.exe
7. Virus.Win32.Alma.37195.exe
8. Virus.Win32.Anuir.3888.exe
9. Virus.Win32.AOC.3676.a.exe
10. Virus.Win32.Apparition.a.exe
11. Virus.Win32.Artelad.2173.exe
12. Virus.Win32.AutoIt.e.exe
13. Virus.Win32.Banka.a.exe
14. Virus.Win32.Belial.2609.exe
15. Virus.Win32.Belus.a.exe
16. Virus.Win32.Blakan.2064.exe
17. Virus.Win32.Bogus.4096.exe
18. Virus.Win32.Bolzano.3164.exe
19. Virus.Win32.Bolzano.4096.f.exe
20. Virus.Win32.Brof.c.exe
21. Virus.Win32.Bube.i.exe

22. Virus.Win32.Bytesv.1442.exe
23. Virus.Win32.Cabanas.Debug.exe
24. Virus.Win32.Cecile.exe
25. Virus.Win32.Celex.a.exe
26. Virus.Win32.Champ.5707.b.exe
27. Virus.Win32.Chiton.a.exe
28. Virus.Win32.Chiton.l.exe
29. Virus.Win32.Chiton.exe
30. Virus.Win32.Compan.a.exe
31. Virus.Win32.Crypto.a.exe
32. Virus.Win32.Darif.a.exe
33. Virus.Win32.Deemo.exe
34. Virus.Win32.Delf.ag.exe
35. Virus.Win32.Delf.bd.exe
36. Virus.Win32.Delf.bm.exe
37. Virus.Win32.Delf.c.exe
38. Virus.Win32.Delf.cq.exe
39. Virus.Win32.Delf.d.exe
40. Virus.Win32.Delf.l.exe
41. Virus.Win32.Dicomp.8192.b.exe
42. Virus.Win32.Ditto.1492.exe
43. Virus.Win32.Doser.4187.exe
44. Virus.Win32.Downloader.a.exe
45. Virus.Win32.Downloader.aj.exe
46. Virus.Win32.Downloader.ar.exe
47. Virus.Win32.Downloader.ba.exe
48. Virus.Win32.Downloader.m.exe
49. Virus.Win32.Downloader.x.exe
50. Virus.Win32.Drol.5337.b.exe
51. Virus.Win32.DunDun.5025.exe
52. Virus.Win32.Egolet.a.exe
53. Virus.Win32.Elkern.dam.exe
54. Virus.Win32.Epoe.1048.exe
55. Virus.Win32.Eva.f.exe
56. Virus.Win32.Evul.8192.b.exe
57. Virus.Win32.Evyl.b.exe
58. Virus.Win32.Expiro.e.exe
59. Virus.Win32.Expiro.n.exe
60. Virus.Win32.Fighter.c.exe
61. Virus.Win32.Folcom.e.exe
62. Virus.Win32.Freebid.exe
63. Virus.Win32.Genu.a.exe
64. Virus.Win32.Ginra.3657.exe
65. Virus.Win32.Giri.5209.exe
66. Virus.Win32.Gremo.2343.exe
67. Virus.Win32.Grum.e.exe

68. Virus.Win32.Grum.n.exe
69. Virus.Win32.Halen.2593.exe
70. Virus.Win32.Hefi.a.exe
71. Virus.Win32.Henky.5668.exe
72. Virus.Win32.Hidrag.a.exe
73. Virus.Win32.HIV.6382.exe
74. Virus.Win32.HLLC.Delfer.e.exe
75. Virus.Win32.HLLC.Hidoc.exe
76. Virus.Win32.HLLC.Sulpex.b.exe
77. Virus.Win32.HLLC.Vedex.exe
78. Virus.Win32.HLLO.Ant.c.exe
79. Virus.Win32.HLLO.Delf.b.exe
80. Virus.Win32.HLLO.Jetto.b.exe
81. Virus.Win32.HLLO.Momac.b.exe
82. Virus.Win32.HLLO.Vogad.exe
83. Virus.Win32.HLLP.Delf.c.exe
84. Virus.Win32.HLLP.Estatic.exe
85. Virus.Win32.HLLP.Flatei.d.exe
86. Virus.Win32.HLLP.Gogo.a.exe
87. Virus.Win32.HLLP.Hetis.exe
88. Virus.Win32.HLLP.Semisoft.e.exe
89. Virus.Win32.HLLP.Semisoft.n.exe
90. Virus.Win32.HLLP.Text.c.exe
91. Virus.Win32.HLLP.VB.d.exe
92. Virus.Win32.HLLP.Werle.a.exe
93. Virus.Win32.HLLP.Zepp.b.exe
94. Virus.Win32.HLLP.Zepp.k.exe
95. Virus.Win32.HLLW.Alcaul.d.exe
96. Virus.Win32.HLLW.Antonio.exe
97. Virus.Win32.HLLW.Azha.exe
98. Virus.Win32.HLLW.Billrus.e.exe
99. Virus.Win32.HLLW.Carpeta.c.exe
100. Virus.Win32.HLLW.Dax.exe
101. Virus.Win32.HLLW.Delf.m.exe
102. Virus.Win32.HLLW.Editorex.exe
103. Virus.Win32.HLLW.Flita.exe
104. Virus.Win32.HLLW.Ghotex.c.exe
105. Virus.Win32.HLLW.Juejue.exe
106. Virus.Win32.HLLW.Lestat.exe
107. Virus.Win32.HLLW.Mintop.b.exe
108. Virus.Win32.HLLW.Munter.a.exe
109. Virus.Win32.HLLW.Oblion.c.exe
110. Virus.Win32.HLLW.Parparo.a.exe
111. Virus.Win32.HLLW.Proget.a.exe
112. Virus.Win32.HLLW.Scareg.exe
113. Virus.Win32.HLLW.SoftSix.b.exe

114. Virus.Win32.HLLW.Tefuss.c.exe
115. Virus.Win32.HLLW.Timese.h.exe
116. Virus.Win32.HLLW.Zule.exe
117. Virus.Win32.Idele.2104.exe
118. Virus.Win32.Iframer.a.exe
119. Virus.Win32.Inrar.a.exe
120. Virus.Win32.Insom.1972.d.exe
121. Virus.Win32.Ipamor.c.exe
122. Virus.Win32.Jeepeg.f.exe
123. Virus.Win32.Jlok.b.exe
124. Virus.Win32.Kangen.a.exe
125. Virus.Win32.Keisan.a.exe
126. Virus.Win32.Kenston.1936.a.exe
127. Virus.Win32.Kies.e.exe
128. Virus.Win32.KMKY.exe
129. Virus.Win32.Kriz.3742.exe
130. Virus.Win32.Kriz.4233.exe
131. Virus.Win32.Ladmar.2004.exe
132. Virus.Win32.Lamebyte.exe
133. Virus.Win32.Lamer.i.exe
134. Virus.Win32.Lamicho.e.exe
135. Virus.Win32.LazyMin.30.exe
136. Virus.Win32.Libertine.d.exe
137. Virus.Win32.Lykov.c.exe
138. Virus.Win32.Magic.7045.c.exe
139. Virus.Win32.Magic.7045.k.exe
140. Virus.Win32.Matrix.909.a.exe
141. Virus.Win32.Matrix.Ordy.e.exe
142. Virus.Win32.Merin.a.exe
143. Virus.Win32.Miam.5110.exe
144. Virus.Win32.Mkar.e.exe
145. Virus.Win32.Mogul.6845.exe
146. Virus.Win32.Mooder.e.exe
147. Virus.Win32.Moontox.a.exe
148. Virus.Win32.Mutarol.2456.exe
149. Virus.Win32.Notime.exe
150. Virus.Win32.Nvrdoc.exe
151. Virus.Win32.Orez.6287.exe
152. Virus.Win32.Peansen.2133.exe
153. Virus.Win32.Pioneer.c.exe
154. Virus.Win32.Porex.c.exe
155. Virus.Win32.Projet.2404.b.exe
156. Virus.Win32.Qozah.3370.exe
157. Virus.Win32.RainSong.4198.exe
158. Virus.Win32.Ramm.f.exe
159. Virus.Win32.Razenya.exe

160. Virus.Win32.Repka.c.exe
161. Virus.Win32.Retroy.a.exe
162. Virus.Win32.Rikenar.1492.exe
163. Virus.Win32.Ruff.4864.exe
164. Virus.Win32.Ryex.exe
165. Virus.Win32.Sality.g.exe
166. Virus.Win32.Sality.q.exe
167. Virus.Win32.Sankei.1766.exe
168. Virus.Win32.Sankei.3480.exe
169. Virus.Win32.Santana.1104.exe
170. Virus.Win32.Savior.1800.exe
171. Virus.Win32.Selfish.h.exe
172. Virus.Win32.Seppuku.1638.exe
173. Virus.Win32.Seppuku.3584.exe
174. Virus.Win32.Sexy.b.exe
175. Virus.Win32.Shodi.g.exe
176. Virus.Win32.Silcer.exe
177. Virus.Win32.Silly.e.exe
178. Virus.Win32.Skoworodka.a.exe
179. Virus.Win32.Slow.8192.b.exe
180. Virus.Win32.Small.1393.exe
181. Virus.Win32.Small.1700.exe
182. Virus.Win32.Small.ap.exe
183. Virus.Win32.Small.t.exe
184. Virus.Win32.Smog.c.exe
185. Virus.Win32.Spit.d.exe
186. Virus.Win32.Stepar.g.exe
187. Virus.Win32.Stupid.a.exe
188. Virus.Win32.Tank.c.exe
189. Virus.Win32.Tenga.b.exe
190. Virus.Win32.Texel.d.exe
191. Virus.Win32.This31.16896.exe
192. Virus.Win32.Thorin.e.exe
193. Virus.Win32.Tirtas.5675.exe
194. Virus.Win32.Towloh.1024.exe
195. Virus.Win32.Trion.c.exe
196. Virus.Win32.Tupac.a.exe
197. Virus.Win32.Ultratt.8167.exe
198. Virus.Win32.Vampiro.7018.exe
199. Virus.Win32.VB.an.exe
200. Virus.Win32.VB.ax.exe
201. Virus.Win32.VB.bf.exe
202. Virus.Win32.VB.bq.exe
203. Virus.Win32.VB.bz.exe
204. Virus.Win32.VB.ch.exe
205. Virus.Win32.VB.cq.exe

206. Virus.Win32.VB.dk.exe
207. Virus.Win32.VB.du.exe
208. Virus.Win32.VB.eh.exe
209. Virus.Win32.VB.eu.exe
210. Virus.Win32.VB.fg.exe
211. Virus.Win32.VB.fr.exe
212. Virus.Win32.VB.gj.exe
213. Virus.Win32.VB.gz.exe
214. Virus.Win32.VB.iu.exe
215. Virus.Win32.VB.jm.exe
216. Virus.Win32.VB.kb.exe
217. Virus.Win32.VB.kk.exe
218. Virus.Win32.VB.kv.exe
219. Virus.Win32.VB.le.exe
220. Virus.Win32.VB.lx.exe
221. Virus.Win32.VB.n.exe
222. Virus.Win32.VB.x.exe
223. Virus.Win32.Vck.3037.exe
224. Virus.Win32.Virut.ab.exe
225. Virus.Win32.Virut.aj.exe
226. Virus.Win32.Virut.au.exe
227. Virus.Win32.Virut.bd.exe
228. Virus.Win32.Virut.bo.exe
229. Virus.Win32.Virut.bx.exe
230. Virus.Win32.Virut.cg.exe
231. Virus.Win32.Virut.t.exe
232. Virus.Win32.Viset.b.exe
233. Virus.Win32.Wanhope.1834.exe
234. Virus.Win32.Weird.e.exe
235. Virus.Win32.Wolf.c.exe
236. Virus.Win32.Xorala.exe
237. Virus.Win32.Xorer.an.exe
238. Virus.Win32.Xorer.bh.exe
239. Virus.Win32.Xorer.cm.exe
240. Virus.Win32.Xorer.cy.exe
241. Virus.Win32.Xorer.eq.exe
242. Virus.Win32.Xorer.ez.exe
243. Virus.Win32.Xorer.fi.exe
244. Virus.Win32.Yerg.9571.exe
245. Virus.Win32.Zakk.a.exe
246. Virus.Win32.Zevity.exe

C Appendix C: System Calls Monitored By The System Calls Recorder

1. NtAcceptConnectPort

2. NtAccessCheck
3. NtAccessCheckAndAuditAlarm
4. NtAccessCheckByType
5. NtAccessCheckByTypeAndAuditAlarm
6. NtAccessCheckByTypeResultList
7. NtAccessCheckByTypeResultListAndAuditAlarm
8. NtAccessCheckByTypeResultListAndAuditAlarmByHandle
9. NtAcquireCMFViewOwnership
10. NtAddAtom
11. NtAddBootEntry
12. NtAddDriverEntry
13. NtAdjustGroupsToken
14. NtAdjustPrivilegesToken
15. NtAlertResumeThread
16. NtAlertThread
17. NtAllocateLocallyUniqueId
18. NtAllocateReserveObject
19. NtAllocateUserPhysicalPages
20. NtAllocateUuids
21. NtAllocateVirtualMemory
22. NtAlpcAcceptConnectPort
23. NtAlpcCancelMessage
24. NtAlpcConnectPort
25. NtAlpcCreatePort
26. NtAlpcCreatePortSection
27. NtAlpcCreateResourceReserve
28. NtAlpcCreateSectionView
29. NtAlpcCreateSecurityContext
30. NtAlpcDeletePortSection
31. NtAlpcDeleteResourceReserve
32. NtAlpcDeleteSectionView
33. NtAlpcDeleteSecurityContext
34. NtAlpcDisconnectPort
35. NtAlpcImpersonateClientOfPort
36. NtAlpcOpenSenderProcess
37. NtAlpcOpenSenderThread
38. NtAlpcQueryInformation
39. NtAlpcQueryInformationMessage
40. NtAlpcRevokeSecurityContext
41. NtAlpcSendWaitReceivePort
42. NtAlpcSetInformation
43. NtApphelpCacheControl
44. NtAreMappedFilesTheSame
45. NtAssignProcessToJobObject
46. NtCallbackReturn
47. NtCancelDeviceWakeupRequest

48. NtCancelIoFile
49. NtCancelIoFileEx
50. NtCancelSynchronousIoFile
51. NtCancelTimer
52. NtClearAllSavepointsTransaction
53. NtClearEvent
54. NtClearSavepointTransaction
55. NtClose
56. NtCloseObjectAuditAlarm
57. NtCommitComplete
58. NtCommitEnlistment
59. NtCommitTransaction
60. NtCompactKeys
61. NtCompareTokens
62. NtCompleteConnectPort
63. NtCompressKey
64. NtConnectPort
65. NtContinue
66. NtCreateChannel
67. NtCreateDebugObject
68. NtCreateDirectoryObject
69. NtCreateEnlistment
70. NtCreateEvent
71. NtCreateEventPair
72. NtCreateFile
73. NtCreateIoCompletion
74. NtCreateJobObject
75. NtCreateJobSet
76. NtCreateKey
77. NtCreateKeyedEvent
78. NtCreateKeyTransacted
79. NtCreateMailslotFile
80. NtCreateMutant
81. NtCreateNamedPipeFile
82. NtCreatePagingFile
83. NtCreatePort
84. NtCreatePrivateNamespace
85. NtCreateProcess
86. NtCreateProcessEx
87. NtCreateProfile
88. NtCreateProfileEx
89. NtCreateResourceManager
90. NtCreateSection
91. NtCreateSemaphore
92. NtCreateSymbolicLinkObject
93. NtCreateThread

94. NtCreateThreadEx
95. NtCreateTimer
96. NtCreateToken
97. NtCreateTransaction
98. NtCreateTransactionManager
99. NtCreateUserProcess
100. NtCreateWaitablePort
101. NtCreateWorkerFactory
102. NtDebugActiveProcess
103. NtDebugContinue
104. NtDelayExecution
105. NtDeleteAtom
106. NtDeleteBootEntry
107. NtDeleteDriverEntry
108. NtDeleteFile
109. NtDeleteKey
110. NtDeleteObjectAuditAlarm
111. NtDeletePrivateNamespace
112. NtDeleteValueKey
113. NtDeviceIoControlFile
114. NtDisableLastKnownGood
115. NtDisplayString
116. NtDrawText
117. NtDuplicateObject
118. NtDuplicateToken
119. NtEnableLastKnownGood
120. NtEnumerateBootEntries
121. NtEnumerateDriverEntries
122. NtEnumerateKey
123. NtEnumerateSystemEnvironmentValuesEx
124. NtEnumerateTransactionObject
125. NtEnumerateValueKey
126. NtExtendSection
127. NtFilterToken
128. NtFindAtom
129. NtFlushBuffersFile
130. NtFlushInstallUILanguage
131. NtFlushInstructionCache
132. NtFlushKey
133. NtFlushProcessWriteBuffers
134. NtFlushVirtualMemory
135. NtFlushWriteBuffer
136. NtFreeUserPhysicalPages
137. NtFreeVirtualMemory
138. NtFreezeRegistry
139. NtFreezeTransactions

140. NtFsControlFile
141. NtGetContextThread
142. NtGetCurrentProcessorNumber
143. NtGetDevicePowerState
144. NtGetMUIRegistryInfo
145. NtGetNextProcess
146. NtGetNextThread
147. NtGetNlsSectionPtr
148. NtGetNotificationResourceManager
149. NtGetPlugPlayEvent
150. NtGetTickCount
151. NtGetWriteWatch
152. NtImpersonateAnonymousToken
153. NtImpersonateClientOfPort
154. NtImpersonateThread
155. NtInitializeNlsFiles
156. NtInitializeRegistry
157. NtInitiatePowerAction
158. NtIsProcessInJob
159. NtIsSystemResumeAutomatic
160. NtIsUILanguageComitted
161. NtListenChannel
162. NtListenPort
163. NtListTransactions
164. NtLoadDriver
165. NtLoadKey
166. NtLoadKey2
167. NtLoadKeyEx
168. NtLockFile
169. NtLockProductActivationKeys
170. NtLockRegistryKey
171. NtLockVirtualMemory
172. NtMakePermanentObject
173. NtMakeTemporaryObject
174. NtMapCMFModule
175. NtMapUserPhysicalPages
176. NtMapUserPhysicalPagesScatter
177. NtMapViewOfSection
178. NtMarshallTransaction
179. NtModifyBootEntry
180. NtModifyDriverEntry
181. NtNotifyChangeDirectoryFile
182. NtNotifyChangeKey
183. NtNotifyChangeMultipleKeys
184. NtNotifyChangeSession
185. NtOpenChannel

186. NtOpenDirectoryObject
187. NtOpenEnlistment
188. NtOpenEvent
189. NtOpenEventPair
190. NtOpenFile
191. NtOpenIoCompletion
192. NtOpenJobObject
193. NtOpenKey
194. NtOpenKeyedEvent
195. NtOpenKeyEx
196. NtOpenKeyTransacted
197. NtOpenKeyTransactedEx
198. NtOpenMutant
199. NtOpenObjectAuditAlarm
200. NtOpenPrivateNamespace
201. NtOpenProcess
202. NtOpenProcessToken
203. NtOpenProcessTokenEx
204. NtOpenResourceManager
205. NtOpenSection
206. NtOpenSemaphore
207. NtOpenSession
208. NtOpenSymbolicLinkObject
209. NtOpenThread
210. NtOpenThreadToken
211. NtOpenThreadTokenEx
212. NtOpenTimer
213. NtOpenTransaction
214. NtOpenTransactionManager
215. NtPlugPlayControl
216. NtPowerInformation
217. NtPrepareComplete
218. NtPrepareEnlistment
219. NtPrePrepareComplete
220. NtPrePrepareEnlistment
221. NtPrivilegeCheck
222. NtPrivilegedServiceAuditAlarm
223. NtPrivilegeObjectAuditAlarm
224. NtPropagationComplete
225. NtPropagationFailed
226. NtProtectVirtualMemory
227. NtPullTransaction
228. NtPulseEvent
229. NtQueryAttributesFile
230. NtQueryBootEntryOrder
231. NtQueryBootOptions

232. NtQueryDebugFilterState
233. NtQueryDefaultLocale
234. NtQueryDefaultUILanguage
235. NtQueryDirectoryFile
236. NtQueryDirectoryObject
237. NtQueryDriverEntryOrder
238. NtQueryEaFile
239. NtQueryEvent
240. NtQueryFullAttributesFile
241. NtQueryInformationAtom
242. NtQueryInformationEnlistment
243. NtQueryInformationFile
244. NtQueryInformationJobObject
245. NtQueryInformationPort
246. NtQueryInformationProcess
247. NtQueryInformationResourceManager
248. NtQueryInformationThread
249. NtQueryInformationToken
250. NtQueryInformationTransaction
251. NtQueryInformationTransactionManager
252. NtQueryInformationWorkerFactory
253. NtQueryInstallUILanguage
254. NtQueryIntervalProfile
255. NtQueryIoCompletion
256. NtQueryKey
257. NtQueryLicenseValue
258. NtQueryMultipleValueKey
259. NtQueryMutant
260. NtQueryObject
261. NtQueryOleDirectoryFile
262. NtQueryOpenSubKeys
263. NtQueryOpenSubKeysEx
264. NtQueryPerformanceCounter
265. NtQueryPortInformationProcess
266. NtQueryQuotaInformationFile
267. NtQuerySection
268. NtQuerySecurityAttributesToken
269. NtQuerySecurityObject
270. NtQuerySemaphore
271. NtQuerySymbolicLinkObject
272. NtQuerySystemEnvironmentValue
273. NtQuerySystemEnvironmentValueEx
274. NtQuerySystemInformation
275. NtQuerySystemInformationEx
276. NtQuerySystemTime
277. NtQueryTimer

278. NtQueryTimerResolution
279. NtQueryValueKey
280. NtQueryVirtualMemory
281. NtQueryVolumeInformationFile
282. NtQueueApcThread
283. NtQueueApcThreadEx
284. NtRaiseException
285. NtRaiseHardError
286. NtReadFile
287. NtReadFileScatter
288. NtReadOnlyEnlistment
289. NtReadRequestData
290. NtReadVirtualMemory
291. NtRecoverEnlistment
292. NtRecoverResourceManager
293. NtRecoverTransactionManager
294. NtRegisterProtocolAddressInformation
295. NtRegisterThreadTerminatePort
296. NtReleaseCMFViewOwnership
297. NtReleaseKeyedEvent
298. NtReleaseMutant
299. NtReleaseSemaphore
300. NtReleaseWorkerFactoryWorker
301. NtRemoveIoCompletion
302. NtRemoveIoCompletionEx
303. NtRemoveProcessDebug
304. NtRenameKey
305. NtRenameTransactionManager
306. NtReplaceKey
307. NtReplacePartitionUnit
308. NtReplyPort
309. NtReplyWaitReceivePort
310. NtReplyWaitReceivePortEx
311. NtReplyWaitReplyPort
312. NtReplyWaitSendChannel
313. NtRequestDeviceWakeup
314. NtRequestPort
315. NtRequestWaitReplyPort
316. NtRequestWakeupLatency
317. NtResetEvent
318. NtResetWriteWatch
319. NtRestoreKey
320. NtResumeProcess
321. NtResumeThread
322. NtRollbackComplete
323. NtRollbackEnlistment

324. NtRollbackSavepointTransaction
325. NtRollbackTransaction
326. NtRollforwardTransactionManager
327. NtSaveKey
328. NtSaveKeyEx
329. NtSaveMergedKeys
330. NtSavepointComplete
331. NtSavepointTransaction
332. NtSecureConnectPort
333. NtSendWaitReplyChannel
334. NtSerializeBoot
335. NtSetBootEntryOrder
336. NtSetBootOptions
337. NtSetContextChannel
338. NtSetContextThread
339. NtSetDebugFilterState
340. NtSetDefaultHardErrorPort
341. NtSetDefaultLocale
342. NtSetDefaultUILanguage
343. NtSetDriverEntryOrder
344. NtSetEaFile
345. NtSetEvent
346. NtSetEventBoostPriority
347. NtSetHighEventPair
348. NtSetHighWaitLowEventPair
349. NtSetHighWaitLowThread
350. NtSetInformationDebugObject
351. NtSetInformationEnlistment
352. NtSetInformationFile
353. NtSetInformationJobObject
354. NtSetInformationKey
355. NtSetInformationObject
356. NtSetInformationProcess
357. NtSetInformationResourceManager
358. NtSetInformationThread
359. NtSetInformationToken
360. NtSetInformationTransaction
361. NtSetInformationTransactionManager
362. NtSetInformationWorkerFactory
363. NtSetIntervalProfile
364. NtSetIoCompletion
365. NtSetIoCompletionEx
366. NtSetLdtEntries
367. NtSetLowEventPair
368. NtSetLowWaitHighEventPair
369. NtSetLowWaitHighThread

370. NtSetQuotaInformationFile
371. NtSetSecurityObject
372. NtSetSystemEnvironmentValue
373. NtSetSystemEnvironmentValueEx
374. NtSetSystemInformation
375. NtSetSystemPowerState
376. NtSetSystemTime
377. NtSetThreadExecutionState
378. NtSetTimer
379. NtSetTimerEx
380. NtSetTimerResolution
381. NtSetUuidSeed
382. NtSetValueKey
383. NtSetVolumeInformationFile
384. NtShutdownSystem
385. NtShutdownWorkerFactory
386. NtSignalAndWaitForSingleObject
387. NtSinglePhaseReject
388. NtStartProfile
389. NtStartTm
390. NtStopProfile
391. NtSuspendProcess
392. NtSuspendThread
393. NtSystemDebugControl
394. NtTerminateJobObject
395. NtTerminateProcess
396. NtTerminateThread
397. NtTestAlert
398. NtThawRegistry
399. NtThawTransactions
400. NtTraceControl
401. NtTraceEvent
402. NtTranslateFilePath
403. NtUmsThreadYield
404. NtUnloadDriver
405. NtUnloadKey
406. NtUnloadKey2
407. NtUnloadKeyEx
408. NtUnlockFile
409. NtUnlockVirtualMemory
410. NtUnmapViewOfSection
411. NtVdmControl
412. NtWaitForDebugEvent
413. NtWaitForKeyedEvent
414. NtWaitForMultipleObjects
415. NtWaitForMultipleObjects32

416. NtWaitForSingleObject
417. NtWaitForWorkViaWorkerFactory
418. NtWaitHighEventPair
419. NtWaitLowEventPair
420. NtWorkerFactoryWorkerReady
421. NtWow64CallFunction64
422. NtWow64CsrAllocateCaptureBuffer
423. NtWow64CsrAllocateMessagePointer
424. NtWow64CsrCaptureMessageBuffer
425. NtWow64CsrCaptureMessageString
426. NtWow64CsrClientCallServer
427. NtWow64CsrClientConnectToServer
428. NtWow64CsrFreeCaptureBuffer
429. NtWow64CsrGetProcessId
430. NtWow64CsrIdentifyAlertableThread
431. NtWow64CsrVerifyRegion
432. NtWow64DebuggerCall
433. NtWow64GetCurrentProcessorNumberEx
434. NtWow64GetNativeSystemInformation
435. NtWow64InterlockedPopEntrySList
436. NtWow64QueryInformationProcess64
437. NtWow64QueryVirtualMemory64
438. NtWow64ReadVirtualMemory64
439. NtWow64WriteVirtualMemory64
440. NtWriteFile
441. NtWriteFileGather
442. NtWriteRequestData
443. NtWriteVirtualMemory
444. NtYieldExecution

D Appendix D: The IDS Configuration File Format

Below is the specification of the ini configuration file used by the IDS. The term *host* relates to the inspecting machine, which should not be damaged in any way by the code inspection.

[host_paths]

trace_output_folder = The path in the host to the folder where the system call recorder output file would be generated

sandbox_path = The path in the host to the sandbox file to load by the *sandbox* instance. For the VM implementation currently used, It is the path to the VMX file of the machine to load.

tracer_executable_folder = The path

tracer_additional_files_folder = C:\Visual Studio 2012\Projects\NtSysCallRecorder

tracer_aux_files_folder = C:\Visual Studio 2012\Projects\NtSysCallRecorder\Microsoft.VC110.CRT

[vm_paths]

trace_output_folder = c:\temp
tracer_folder = c:\temp
shell_path = C:\windows\system32\cmd.exe
[vm_settings]
clean_snapshot_name = Snapshot after update
vm_user_name = Administrator
vm_password = 12345678
[tracer_settings]
executable_name = NtSysCallRecorder.exe
additional_files = dbgCopy.dll,NtSysCallRecorder.cfg
aux_files = msvcp110.dll,msvcr110.dll,vccorlib110.dll

E Appendix E: The Windows-specific System Calls Recorder Output Example

An example the output of our system calls recorder for the application calc.exe is shown below. Only the first 8 system calls are shown:

Process 1400 starting at 01012475

C:\WINDOWS\system32\calc.exe

Loaded DLL at 7C900000 ntdll.dll

NtOpenKey(KeyHandle=0x7fc74, DesiredAccess=GENERIC_READ, ObjectAttributes="\Registry\Machine\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\calc.exe") ⇒ 0xc0000034 [2 'The system cannot find the file specified.']

NtOpenKey(KeyHandle=0x7fc44 [0x7fc], DesiredAccess=GENERIC_READ, ObjectAttributes="\Registry\Machine\System\CurrentControlSet\Control\Session Manager") ⇒ 0

NtQueryValueKey(KeyHandle=0x7fc, ValueName="CWDIllegalInDLLSearch", KeyValueInformationClass=2 [KeyValuePartialInformation], KeyValueInformation=0x7f7f0, Length=0x400, ResultLength=0x7fbf8) ⇒ 0xc0000034 [2 'The system cannot find the file specified.']

NtClose(Handle=0x7fc) ⇒ 0

NtOpenKeyedEvent(KeyedEventHandle=0x7fb14 [0x7fc], DesiredAccess=MAXIMUM_ALLOWED, ObjectAttributes="\KernelObjects\CritSecOutOfMemoryEvent") ⇒ 0

NtQuerySystemInformation(SystemInformationClass=0 [SystemBasicInformation], SystemInformation=0x7fa50, Length=0x2c, ReturnLength=null) ⇒ 0

NtQuerySystemInformation(SystemInformationClass=0 [SystemBasicInformation], SystemInformation=0x7f928, Length=0x2c, ReturnLength=null) ⇒ 0

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0x7f9c0 [0x000a0000], ZeroBits=0, pSize=0x7f9ec [0x00100000], flAllocationType=0x2000, flProtect=4) ⇒ 0

more system calls appear here

F Appendix F: The Platform-Independent (Normalized) System Calls Recorder Output Example

An example the output of our system calls recorder from the previous appendix after it was converted by our parser is shown below. Only the first 8 system calls are shown:

<?xml version="1.0"?>
<process_trace pid="1400" path="C:\WINDOWS\system32\calc.exe">
<system_calls>
<system_call return_value="0xc0000034 [2 'The system cannot find the file specified.']" name="NtOpenKey">
<arg name="KeyHandle" value="0x7fc74"/>
<arg name="DesiredAccess" value="GENERIC_READ"/>
<arg name="ObjectAttributes" value=""\Registry\Machine\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\calc.exe""/>
</system_call>
<system_call return_value="0" name="NtOpenKey">
<arg name="KeyHandle" value="0x7fc44 [0x7fc]"/>
<arg name="DesiredAccess" value="GENERIC_READ"/>
<arg name="ObjectAttributes" value=
""\Registry\Machine\System\CurrentControlSet\Control\Session Manager""/>
</system_call>
<system_call return_value="0xc0000034 [2 'The system cannot find the file specified.']" name="NtQueryValueKey">
<arg name="KeyHandle" value="0x7fc"/>
<arg name="ValueName" value=""CWDIllegalInDLLSearch""/>
<arg name="KeyValueInformationClass" value="2 [KeyValuePartialInformation]"/>
<arg name="KeyValueInformation" value="0x7f7f0"/>
<arg name="Length" value="0x400"/>
<arg name="ResultLength" value="0x7fbf8"/>
</system_call>
<system_call return_value="0" name="NtClose">
<arg name="Handle" value="0x7fc"/>
</system_call>
<system_call return_value="0" name="NtOpenKeyedEvent">
<arg name="KeyedEventHandle" value="0x7fb14 [0x7fc]"/>
<arg name="DesiredAccess" value="MAXIMUM_ALLOWED"/>
<arg name="ObjectAttributes" value=""\KernelObjects\CritSecOutOfMemoryEvent""/>
</system_call>
<system_call return_value="0" name="NtQuerySystemInformation">
<arg name="SystemInformationClass" value="0 [SystemBasicInformation]"/>
<arg name="SystemInformation" value="0x7fa50"/>
<arg name="Length" value="0x2c"/>
<arg name="ReturnLength" value="null"/>
</system_call>

```
    <system_call return_value="0" name="NtQuerySystemInformation">
     <arg name="SystemInformationClass" value="0 [SystemBasicInformation]"/>
        <arg name="SystemInformation" value="0x7f928"/>
        <arg name="Length" value="0x2c"/>
        <arg name="ReturnLength" value="null"/>
    </system_call>
    <system_call return_value="0" name="NtAllocateVirtualMemory">
        <arg name="ProcessHandle" value="-1"/>
        <arg name="lpAddress" value="0x7f9c0 [0x000a0000]"/>
        <arg name="ZeroBits" value="0"/>
        <arg name="pSize" value="0x7f9ec [0x00100000]"/>
        <arg name="flAllocationType" value="0x2000"/>
        <arg name="flProtect" value="4"/>
    </system_call>
# more system calls appear here
</system_calls>
</process_trace>
```