Colored Petri Nets as the Enabling Technology in Intrusion Detection Systems

A. Dolgikh, T. Nykodym, V. Skormin,
Binghamton University, Binghamton, NY
adolgik1@binghamton.edu

J. Antonakos,
Broome Community College, Binghamton, NY

M. Baimukhamedov,
Kostanai Technical University, Kostanai, Kazakhstan

***Abstract*—Behavior based intrusion detection technologies are increasingly popular. Traditionally behavior patterns are expressed as specific signatures defined in the system call domain. This approach has various drawbacks and is vulnerable to possible obfuscations.**

The IDS approach discussed herein addresses process behavior in terms of functionalities, i.e. particular process objectives. The functionalities are formalized in the form that is independent of their specific realizations and is obfuscation resistant. The malware is detected by particular sets of functionalities exposed by programs during their execution. The approach implies the selection of common malicious functionalities, followed by formal description of these functionalities via specific system call combinations. In the detection domain, monitored system calls are combined into API functions utilizing Colored Petri nets (CPN). After that API functions are combined into malicious functionalities, indicative of malware attack, also using CPN. The advantages of CPN utilization for dynamic code analysis are described.

By its nature the described approach is signature-based. The CPN technology is the backbone of the described approach: CPNs are used to define the functionalities of interests as behavior signatures, and at the same time serve as the mechanism for the signature detection. The paper describes a unique general-purpose software tool implementing CPN. It constitutes the enabling technology for the described IDS approach, and has many additional applications for modeling and monitoring complex hierarchical systems of discrete events.

***Keywords*—behavior based IDS, Colored Petri Net, functionality detection, behavior detection**

I. INTRODUCTION

Modern malware uses increasingly advanced schemes to avoid detection. Nowadays some malware is crafted for one time use against high value targets [1]. This type of malware is very hard to detect since it employs advanced mimicry and hiding techniques. In addition, it often incorporates insider knowledge in order to hide and spread efficiently. The attack usually evolves slowly over time with a small footprint making it even harder to notice using statistical methods [2].

System protection is a popular research area. Risk analysis, vulnerability analysis, intrusion detection systems and forensics systems are being researched and designed. The subarea of intrusion detection features many approaches of varying efficiency and dependability [3].

This research was funded by the Air Force Office of Scientific Research (AFOSR).

In the domain of behavioral intrusion detection systems (IDS), most notable recent efforts are [4, 5, 6, 7]. However, their general weakness stems from the lack of clear and easy to understand technology for the specification and detection of malicious operations. Additional drawbacks result from the limited monitoring time window and wasted system resources for tracking repeated operations (Read/Write operations, virtual memory operations, etc.) making them vulnerable to DoS attacks. Behavior based intrusion detection systems utilizing Colored Petri nets as a tracking mechanism do not have such drawbacks. They can monitor large volumes of operations over a long period of time. This renders slow or high volume attacks impractical.

II. COLORED PETRI NET DESCRIPTION

Colored Petri nets (CPN) are a backward compatible extension of Petri Nets [8]. A CPN consists of places, transitions and arcs. Each place may contain a dynamically varying number of tokens. Each token in turn may contain an attached data value called token color. The data value can have any complex structure (in some cases even another CPN). All tokens belonging to a particular place must have colors corresponding to the type of the place. This type is also called a color set of the place.

One of the most useful properties of Petri Nets is that they have intuitive visual representation. Places are represented by circles or ovals, transitions by rectangles, and tokens by dots in the places. Each place, transition or arc can carry a rich set of inscriptions. It makes the task of creating and understanding a CPN much easier. In spite of the simplicity, Colored Petri Nets constitute a very powerful tool. CPNs have found a lot of applications in industrial engineering [9], military planning [10] and software engineering [11], [12]. In addition to the applications mentioned above, Colored Petri Nets provide an efficient framework for building event detection systems. The inherent parallelism of CPNs perfectly matches the real world environments when most events occur in parallel.

To illustrate some of the basic concepts of a CPN consider the CPN in Figure 1. There are three places (Sensitive Data Accessed, Communication Requested, Leak) and one transition (the transition guard expression compares process IDs). The color set of the place “Sensitive Data Accessed” is positive integers. The color sets of the places “Communication Requested” and “Leak” are products of sets of positive integers and strings.

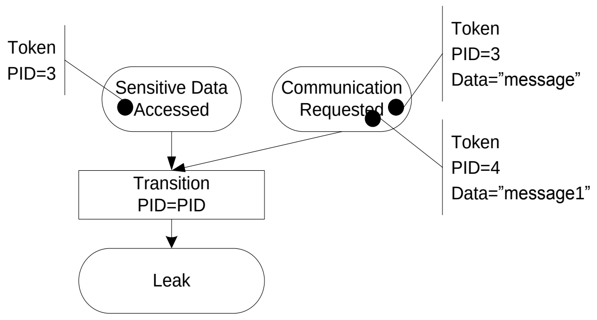

Figure 1 Sensitive data encryption decision making (step 1)

The distribution of tokens over the places is called marking. For the marking presented in Figure 1 the transition is enabled by tokens with PID=3. This means that transition may occur and move the CPN into the next marking by removing tokens with PID=3 from places "Sensitive Data Accessed" and "Communication Requested" followed by placing of a token into the "Leak" place (see Figure 2). In the resulting marking the CPN has tokens in places "Communication Requested" and "Leak".

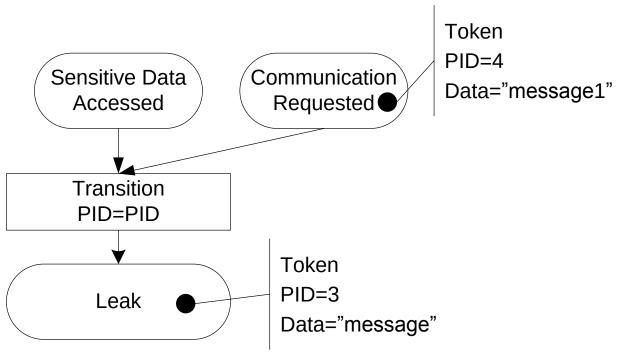

Figure 2 Sensitive data encryption decision making (step 2)

In the resulting state the token residing in the "Leak" place reflects the following facts:

- The program has accessed sensitive data
- The same program requested communication

This constitutes a complex event which by itself means more than a mere combination of events it consists of. This event indicates the possibility of sensitive data leak from the system. Therefore, the CPN can serve as an event detection system capable of bridging the semantic gap between the low level and high level perceptions of the same abstract process. The "Communication Requested" or "Sensitive Data Accessed" places could receive tokens from other parts of a larger CPN making "Leak" a complex hierarchical event composed of several simpler events (see Figure 3)

CPNs can assemble such hierarchical events very efficiently from the viewpoint of memory requirements and processing. Storing a token does not take up much memory space. Computation of guard and arc expressions is strictly local (involves only neighboring nodes). Therefore it can be done in parallel on modern multicore CPUs. In addition, the detection workload can be easily distributed among a number of disjoint CPNs placed in the network hosts.

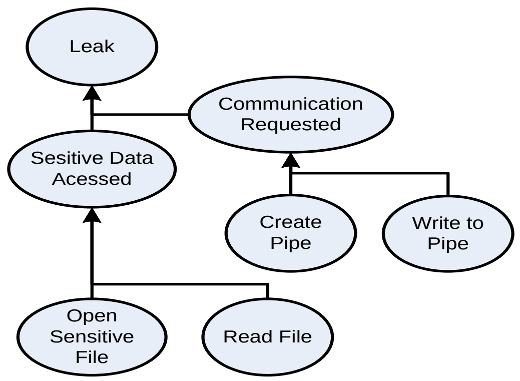

Figure 3 Event hierarchy

III. SEMANTIC APPROACH

The main goal of behavior based IDS is to identify and report malicious operations over the resources of the system. This might include but is not limited to self-distribution without user's consent, information leaking (which can be legitimate for some authorized programs), malicious file hosting (which can be legitimate depending on the environment), etc. As one may see it is hard to define and even harder to detect malicious behavior. The novel IDS approach we proposed [13] does not classify program behavior as malicious or benign. The detection of malicious behavior is replaced by the detection of specific functionalities (behavioral patterns) and later tagging some functionalities or sets of functionalities as malicious in the current environment.

The functionality can be understood as a set of operations achieving tangible results in the program's environment. In most cases in order to change the environment or communicate with the outside world the program issues system calls to an operating system (see Figure 4). Therefore system call monitoring is the ideal place for tracing program behavior. System call monitoring provides a reliable and in many cases incorruptible source of raw behavior-carrying data for the functionality recognition.

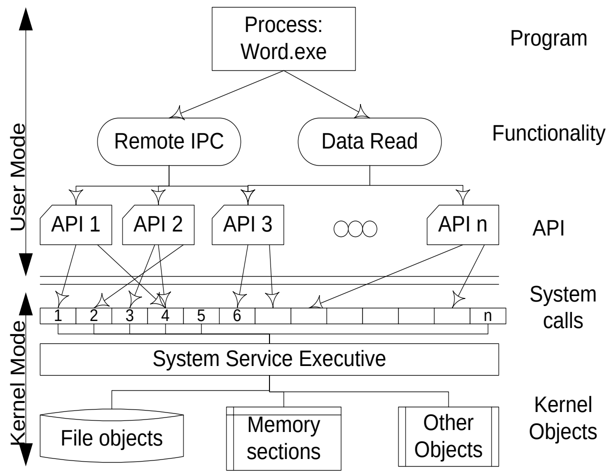

Figure 4 Program functionality implementation

Although implemented as a set of system calls, a functionality is not clearly defined by the programs or programmers. Therefore the semantic gap exists between the functionality level of the program and its system call level implementation.

We suggest a natural language analogy as pictured in Figure 5. Sentences carrying the essence of what the speaker meant to say correspond to functionalities, words correspond to APIs and letters correspond to system calls. It is easy to see how the semantics are being lost along the path from sentences to letters. The famous sentence is barely recognizable behind the set of letters presented at the left column.

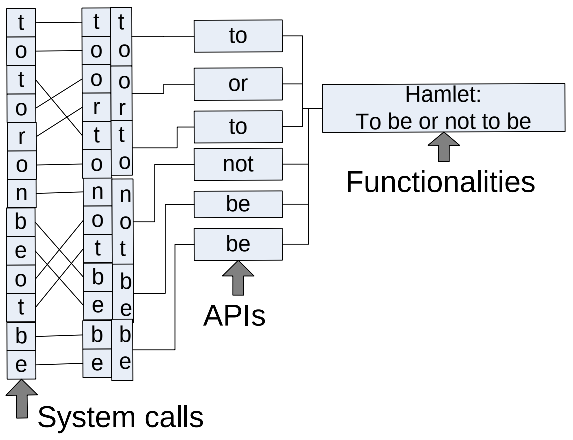
t o t o r o n b e o t b e
t o o r t o n o t b e b e
t o o r t o n o t b e b e
to
or
to
not
be
be
Hamlet:
To be or not to be
Functionalities
APIs
System calls

Figure 5 Natural language analogy with functionality assembly

To bridge the gap between the low level and high level view of the system a mechanism capable of detecting hierarchical events with multiple links is needed. The appropriate candidate for this type of effort is the Colored Petri Net. As it was pointed out in the previous section, CPNs can efficiently assemble complex events from the stream of low level events.

A. Functionality specification

In the paper [13] the authors described the specification of functionalities through the use of UML activity diagrams (AD) (see Figure 6). This specification method allows an expert to define functionalities using an easy to comprehend graphic language. Later those specifications can be automatically converted into the Colored Petri Net specifications by the use of the AD-to-CPN compiler which is currently being developed by the authors.

The AD specification provided by an expert does not always cover every possible implementation of the functionality in question. This peculiarity can be exploited by an attacker. An attacker may create an obfuscation tool capable of modifying malware to implement/execute certain operations in a different way each time it is used.

In order to make the development of automatic obfuscation engines problematic, an AD-to-CPN compiler automatically generalizes the specification provided by an expert. The generalization process detects operations over OS objects (files, IPC, registry, processes) in the activity diagram specified by an expert. Then it generalizes the activity diagram undermining a list of imaginable (as of now) obfuscations. The list of detectable obfuscations includes: partitioning of the operation among several processes; renaming, moving, copying of the resources and indirect access to the resources (symbolic/hard links, reparse points, handle duplication). The same anti-obfuscation process can be applied to CPN signatures effectively by application of hierarchical CPNs and relaxing constraints on intraprocess monitoring.

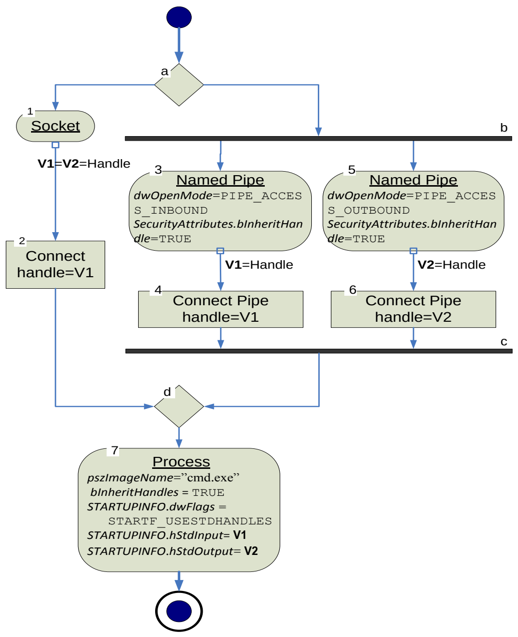
a
b
1 Socket
V1=V2=Handle
2 Connect handle=V1
3 Named Pipe
dwOpenMode=PIPE_ACCESS_INBOUND
SecurityAttributes.bInheritHandle=TRUE
5 Named Pipe
dwOpenMode=PIPE_ACCESS_OUTBOUND
SecurityAttributes.bInheritHandle=TRUE
V1=Handle
V2=Handle
4 Connect Pipe handle=V1
6 Connect Pipe handle=V2
c
d
7 Process
pszImageName="cmd.exe"
bInheritHandles = TRUE
STARTUPINFO.dwFlags = STARTF_USESTDHANDLES
STARTUPINFO.hStdInput= V1
STARTUPINFO.hStdOutput= V2

Figure 6 Example of activity diagram specification for bind/reverse shell

The resulting CPN specification is able to detect the original functionality defined by an expert and the much richer set of obfuscated functionalities. As a result the process of creation of new malware becomes harder to automate which in turn results in substantially increased development costs for cybercriminals.

B. Functionality detection

The functionality specified in CPN format can be detected by the direct simulation of the corresponding CPN utilizing input events coming from the system call monitor.

The mechanics of the functionality recognition can be fully demonstrated by a simple enough example of the detection of the “drop & execute” functionality. The CPN based signature capable of detecting this functionality is featured in Figure 7. Note that this CPN signature does not feature anti-obfuscation schematics or cleanup schematics for the clarity of presentation.

Assume that initially the program executes the following sequence of system calls:

```
ZwCreateFile("temp.exe") returning handle=1;
ZwWriteFile(handle=1, "Data 1");
ZwWriteFile(handle=2, "Data 2");
```

It will result in the marking of the CPN presented in Figure 7.

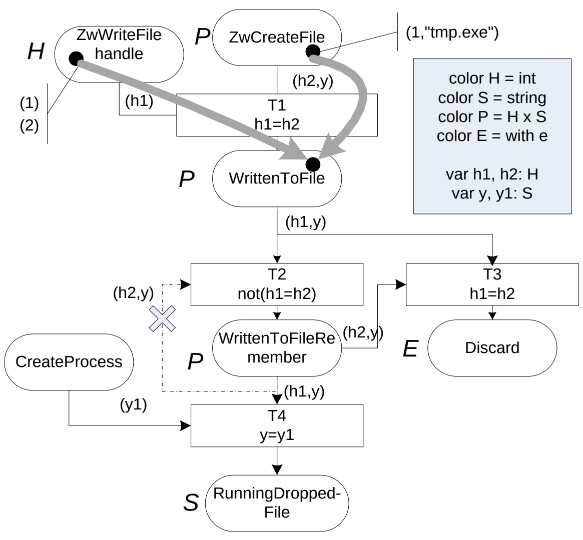

Figure 7 Sample CPN detecting "drop & execute" functionality

After the transition T1 occurs for tokens (1) and (1, "tmp.exe") these tokens are removed from the ZwWriteFile and ZwCreateFile places followed by firing a token into the WrittenToFile place. Execution of additional write calls

```
ZwWriteFile(handle=1, "Data 3");
…;
ZwWriteFile(handle=1, "Data n");
```

leads to firing one token to the WriteRemeber place by transition T2. All remaining tokens are removed and discarded by the occurrence of transition T3 (Figure 8):

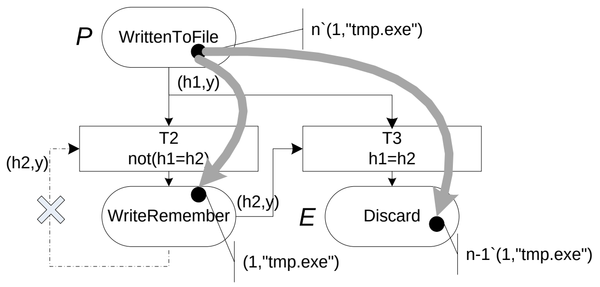

Figure 8 Sample CPN simulation step 2

The completion of the CreateProcess ("tmp.exe") functionality results in the occurrence of transition T4 which in turn fires a token into the place RunningDroppedFile (see Figure 9).

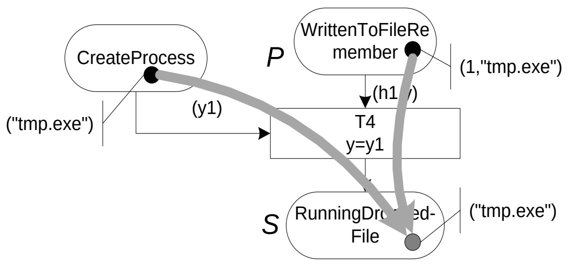

Figure 9 Sample CPN simulation step 3

The arrival of a token to the place RunningDroppedFile can be interpreted as the recognition of the corresponding functionality "drop & execute". At this point some response action can be carried out such as: simply increase functionality counter, send a report to the monitoring system, fire a token to the place of the CPN of higher order in hierarchy, etc.

As it was noted in paragraph II, a token can be made to carry all event assembly information along. For the token arriving at the RunningDroppedFile place, the back trace featured in Figure 10 can be obtained. This back trace was generated by the APIMon monitoring software developed by authors of this paper.

```
Place running_dropped_file(1)
|---Place Mirror_of_write_to_executable=
|  |---Place Mirror_of_write_remember_transfers_unique=
|     |---Place Joined_create_&_open=
|     |  |---CallName = ZwCreateFile
|     |  |---DesiredAccess =
|     |  |       FILE_READ_ATTRIBUTES, SYNCHRONIZE, GENERIC_WRITE,
|     |  |       GENERIC_READ
|     |  |---FileCreationFlags=
|     |  |       FILE_SYNCHRONOUS_IO_NONALERT, FILE_NON_DIRECTORY_FILE
|     |  |---FileHandle = 708
|     |  |---ObjectName = \??\C:\...\Temp\wmbose.exe
|     |---Place ZwWriteFile=
|        |---CallName = ntdll.dll.ZwWriteFile
|        |---buffer = MZ@<skipped>
|        |---BytesWritten = 120708
|        |---FileHandle = 708
|---Place Mirror_of_CreateProcess=
   |---CallName = CreateProcess
   |---ApplicationName = C:\...\Temp\wmbose.exe
   |---CommandLine = "C:\...\Temp\wmbose.exe"
   |---FirstThreadHandle = 964
   |---FirstThreadId = 3996
   |---PID = 3332
   |---ProcessCreationFlags =
   |       CREATE_SUSPENDED, CREATE_NEW_CONSOLE,
   |       CREATE_UNICODE_ENVIRONMENT, CREATE_DEFAULT_ERROR_MODE
   |---ProcessHandle = 960
   |---ProcessId = 3988
   |---std_ERR_Handle = 0
   |---std_IN_Handle = 0
   |---std_OUT_Handle = 0
```

Picture 10 Detection backtrace for drop & execute functionality

The ability of CPNs to assemble complex events from more simple ones turns out to be very useful in the event of a security breach. High level events observed by the CPN allow the security researcher to operate with more meaningful information than before. It results in increased productivity and efficiency.

IV. PROGRAM DESCRIPTION

The detection of functionalities can be done straight from the system calls using one large CPN. It allows for sharing parts of the CPN structure between different functionalities. It may seem beneficial since it might help to reduce the overall number of CPN nodes. Unfortunately it also makes it difficult to understand and debug the results.

A much more scalable approach is to perform the detection in relatively independent CPNs in a number of separate stages. The APIMon monitoring software does it in correspondence to the multilayer structure of commodity software. At each layer some part of the semantic gap between the low level system view and program functionalities is bridged.

At the first layer system calls are intercepted and sent to high performance low level CPNs (see Figure 11). The main function of low level CPNs is to reliably rebuild API calls from the stream of system calls. The CPNs at this level are hard coded in C++ for efficient processing. After that stage, recognized API calls are sent for further analysis to the high level CPNs (see Figure 12).

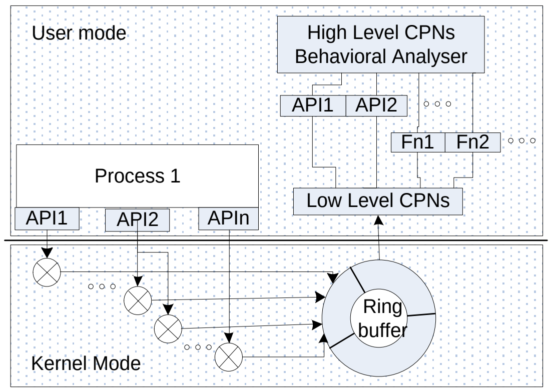

Figure 11 Low level event processing structure

High level CPNs are able to perform arbitrary complex functionality recognition. Anti-obfuscation techniques can be applied at this level. High level CPNs are executed within a configurable CPN engine. It is implemented in C#. It abstracts all internal semantics of Petri Nets simulation and gives the CPN developer an opportunity to effectively express behavior of programs in his subject area.

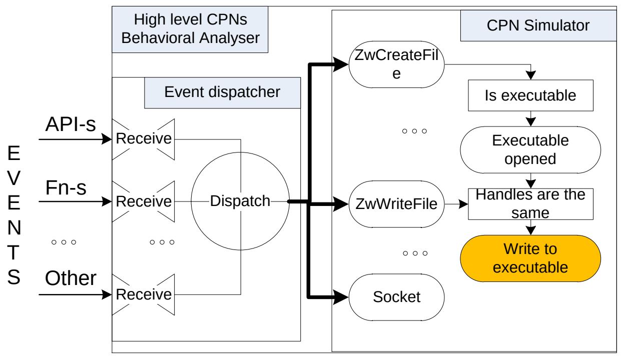

Figure 12 High level behavior processing engine

It is worth noting, that any CPN specification is twofold. On one hand it is structural specification and on another inscriptional. In our system, structural specification is built upon three generic elements: place, transition, and arc. In order to create the CPN structure, the developer simply connects places and transitions by using arcs. After this is accomplished, inscriptions can be introduced into the CPN by specifying arbitrary expressions for arcs or guard expressions for transitions. Expressions are written in C# LINQ. This results in readable and, at the same time, execution efficient specification.

V. CONCLUSION

High level monitoring of software is usually intractable, inaccessible or insecure. Consequently it is beneficial to monitor lower level events and restore original events by means of using some event detection system.

Colored Petri Nets provide superb detection intelligence for this type of task. Using the right CPN signatures it is possible to efficiently reconstruct high level system events from the stream of system calls. The reconstruction can be done in several stages, closing the semantic gap in smaller steps. Application of CPNs results not only in the recognition of particular functionalities, but the whole set of technologies start to be available: anti-obfuscation tracking, structured SIEM reports, and unknown program classification. Some of these technologies were explained in this paper.

The CPN based engine developed by authors proved to be an enabling technology for ongoing and new research.

VI. ACKNOWLEDGMENTS

This research was funded by the Air Force Office of Scientific Research (AFOSR). The authors are grateful to Dr. Robert Herklotz of AFOSR for supporting this effort and to Dr. A. Tokhtabayev of George Mason University for valuable suggestions and friendly discussions.

VII. REFERENCES

1. A. Matrosov, E. Rodionov, D. Harley, J. Malcho, Stuxnet Under the Microscope, Revision 1.31, ESET, January 2011
2. C.Kruegel, E.Kirda, D.Mutz, W. Robertson, G.VignaAutomating mimicry attacks using static binary analysis. USENIX Security Symposium - Volume 14 (SSYM'05).
3. S. Axelsson. Intrusion detection systems: A survey and taxonomy. Technical Report 99-15, Department of Computer Engineering, Chalmers University, March 2000
4. M. Fredrikson, M. Christodorescu, S. Jha, Dynamic Behavior Matching: A ComplexityAnalysis and New Approximation Algorithms, To appear in the 2011 Conference on Automated Deduction
5. M. Fredrikson, M. Christodorescu, S. Jha, R. Sailer, and X. Yang. Synthesizing near optimal specifications of malicious behavior. In Proceedings of the IEEE Symposium of Security and Privacy, 2010.
6. Y. Park, D. Reeves, V. Mulukutla, and B. Sundaravel. Fast malware classification by automated behavioral graph matching. In Proceedings of the Workshop on Cyber Security and Information Intelligence Research, 2010.
7. C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. K. Xiaoyong Zhou, X. Wang. Efficient and effective malware detection at the end host. In Proceedings of the Usenix Security Symposium, 2009.
8. Kurt Jensen. "Coloured Petri nets (2nd ed.): basic concepts, analysis methods and practical use", volume 1, Springer-Verlag, Berlin, 1996
9. L. Jansen, M. Hörste, E. Schnieder: Technical Issues in Modelling the European Train Control System. Proceedings of the Workshop on Practical Use of Coloured Petri Nets and Design/CPN, Aarhus 1998
10. L.Kristensen, P. Mechlenborg, L. Zhang, B. Mitchell, G. Gallash: Model-based Development of a Course of Action Scheduling Tool. In International Journal on Software Tools for Technology Transfer, 10(1), 2008, Springer-Verlag,
11. K. Zurowska and R. Deters:Overcoming Failures in Composite Web Services by Analysing Colored Petri Nets: Proceedings of the Eighth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, October 2007, Department of Computer Science, University of Aarhus, PB-584, 87-106
12. G.Helmer, J. Wong, M.Slagell, V.Honavar, L. Miller, Y. Wang, X. Wang, N.Stakhanova "A Software Fault Tree and Colored Petri Nets based specification, Design and Implementation of Agent Based Intrusion Detection Systems". International Journal of Information and Computer Security, Vol. 1, no 1/2, pp. 109-142, 2007
13. A. Tokhtabayev, V. Skormin, A. Dolgikh. Expressive, efficient and obfuscation resilient behavior based IDS. In Proceedings of the 15th European conference on Research in computer security (ESORICS'10)