Security Avoidance in a Windows Application using Dynamic Instrumentation

Sriya Santhanam and Janani Thanigachalam

Department of Computer Sciences

University of Wisconsin, Madison

{sriya,janani}@cs.wisc.edu

Abstract

Numerous commodity Windows-based applications are available for use as free evaluation copies for limited periods of time. These applications constitute prime candidates for security attacks and we use the dynamic instrumentation capabilities provided by the DynInst API to demonstrate one such attack. The attack methodology uses program inspection and runtime code modification, and is applicable to both stripped and unstripped Windows binaries. While our attack strategy is generic to time-limited trial applications that use the Microsoft C runtime library, our chosen target application is the 30-day trial version of SecureCRT for Windows. Other applications attacked using our tool include VShell Server, CRT, AbsoluteFTP and EnTunnel.

1 Introduction

The goal of this effort is to demonstrate the use of dynamic code instrumentation technology to subvert software security in a commercial application and thereby draw notice among the software security community to its potential in the domain of attacks. Our project is based on the DynInst API, which provides a C++ class library to instrument and modify application programs during execution [1]. Through this project we demonstrate the implementation of this API on Windows, one platform in which it is yet to undergo rigorous testing. We also utilized recent functionality added to the DynInst API to handle stripped binary code [5]. Stripped binary code is code that is either partially or completely devoid of symbol table information and is hence more difficult to trace. Most Windows-based commercial applications use stripped binary code owing to its reduced size. Moreover, Windows compilers tend to generate stripped binary code by default, with debug information preserved in a separate file. Our demonstration thus targets the more common case by

being applicable to both partial and fully stripped executables. A huge class of such Windows applications are available as free downloads from popular sites like download.com, coffeecub.com and sofotex.com among others.

Several popular commercial applications offer fully functional trial versions as free downloads. These trial applications use some form of validation to ensure licensed purchase of software for use beyond the trial period. Such validity checks can either be performed locally, like checking for a license file or a valid trial date, or remotely by contacting a license server. We targeted one such application that uses local license checking, the trial version of SecureCRT v 4.1.9, and performed runtime modification of its code using the DynInst API to extend its use beyond the trial period. We also demonstrate the applicability of the attack to other time-based trial applications, including VShell Server, CRT, AbsoluteFTP and EnTunnel.

The rest of this paper is organized as follows. Related effort in the area of program instrumentation and software security is discussed in Section 2.The DynInst API and the program environment are discussed in Section 3 while Section 4 elaborates on the implementation details of the attack. Suggested countermeasures to protect applications from such attacks are presented in Section 5 while ideas for future work are mentioned in Section 6. Concluding remarks are presented in the final section.

2 Related Work

The domain of program instrumentation has been widely explored [7] with the implementation of various tools and libraries that enable both static and dynamic program instrumentation at the bytecode [8] and binary levels. Some instances of static instrumentation libraries include EEL [9], ATOM [10] and Etch [11] while dynamic instrumentation libraries include the DynInst API [2], PIN [12] and DELI [13]. Our work is based on the instrumentation capabilities of the DynInst API and uses it to analyze program execution sequences, dynamically load custom libraries into the process' address space and replace function calls at runtime.

Dynamic code instrumentation is a technology that has been used to build a wide range of applications ranging from performance monitoring tools and dynamic code optimizers to program inspection tools and software security checkers. Some examples of such applications include the Paradyn Parallel Performance Measurement Tool [21], a dynamic optimization system called Dynamo [14], IBM's Jikes optimizer [15], and debuggers like FULLDOC [16], iWatcher [17] and GDB [18].

Binary instrumentation technology has also had a significant impact on the domain of software security. Program shepherding [19] is a means for monitoring the control flow of an executing program to enforce security policies while Safe virtual execution (SVE) [20] allows the running of untrusted programs on a

host system by enabling the ability for the host to control resource utilization through software dynamic translation (SDT) techniques.

On the other side of the spectrum, the application of dynamic instrumentation techniques in the area of security avoidance has been demonstrated using the same API in a project by Prof. Barton Miller and his students [4]. This effort involved analyzing unencrypted, unstripped binary code (with symbol information intact) on a Linux system to detect and bypass remote calls made by a target application, Adobe Framemaker, to a license server. Our work involved instrumenting a Windows application. Since a large complement of proprietary software operate on a Windows environment, we believe this to be a useful demonstration of a security attack based on dynamic code synthesis. Our target application, SecureCRT, uses local information to perform a validity check as opposed to contacting a remote server. Further, our attack is applicable to a stripped binary executable and uses recent functionality added to the DynInst API [5]. Since most proprietary Windows-based applications use stripped binary code owing to its reduced size and relative robustness to such binary instrumentation attacks, we believe our demonstration addresses a wider software set. Moreover, although information about several attacks on Windows-based software licenses are popularly available on the Internet, we are yet to find published work that uses dynamic instrumentation to achieve similar results.

3 Environment

3.1 The DynInst API

DynInst is a machine-independent binary rewriting library [1] that can be used to insert snippets of code and make modifications to the behavior of a running program. We can use this library to trace the control flow of an executing program and recreate function call graphs. The API also permits changing or removing subroutine calls in the application program and enables user-defined libraries to be loaded into the application's address space. DynInst has been used in a variety of applications like debuggers and performance monitoring tools as well as for utilities such as process checkpointing [21, 22]. For the purposes of our project, we have worked with release 4.1.1 of the DynInst library on a Windows XP platform.

Our project environment is composed of two components: a mutator and a mutatee. The target application that we instrument is called the mutatee and the controlling program is called the mutator. The mutator uses DynInst functionality to insert or modify code in the mutatee. DynInst can attach to the mutatee if the process is already running or can start the mutatee as a fresh process [3]. We observed that by executing the application using DynInst, there was no change in its behavior although the initial instrumentation causes the application to start up a bit slowly.

3.2 The Application

We wished to target a fully functional trial version of a popular commercial Windows application that expires either within a certain time period or after a certain number of trials. Some examples of such applications include SecureCRT, Adobe Photoshop, Macromedia Flash Player, XWin-32 and Winzip. Our chosen target application for this project is the free trial version of SecureCRT, an SSH client for Windows that is a product of Van Dykes Software [23]. This application is available as a free download for a limited period of 30 days and uses local checks to test for the expiry of the trial period.

4 Implementation

Time-based trial applications often use system calls defined in standard libraries to obtain the current system date and time and compare it with the installation date to check for product expiry. Our attack is aimed at extending the accessibility of such a time-limited trial application indefinitely beyond the trial period. In doing so, we instrument a function call in the module containing the Microsoft C Runtime Library imported by the application, rather than instrumenting any application-specific function. Hence, our approach is generic and portable to the large number of trial applications that use this library.

4.1 Attack Strategy

The target application is written in C and uses standard Windows libraries, including the Microsoft C runtime library and the Microsoft Foundation Classes application framework. Our initial step was to analyze the application, obtaining a list of all the modules and procedures that could potentially be used during execution. Since we were dealing with a partially stripped binary, i.e. with symbol table information removed, the majority of the function names in the modules of the application were visible only as hexadecimal values representing the base address of the function in memory.

Our next step was to analyze the execution sequence of the application and we did so by creating a dynamic linked library (DLL) with some user-defined print functions, loading this library into the application's address space and instrumenting each function in the application image to call the print functions from our library upon entry and exit. Since we could not instrument all the 27 modules loaded by the application over the course of a single run owing to heap overflow issues, we instrumented the functions on a per-module basis, changing the instrumented module for each run of the application. Eventually, we were able to reconstruct a call graph for the application. Some system libraries were not compatible with the instrumentation and hence functions called from these libraries were missing from our call graph. However, we were particularly interested in analyzing the functions in the module `License40.dll` that was packaged along with the appli-

cation but noticed that this module never showed up in our analysis of the application image. Recognizing this module to be a library that was loaded at a later point during program execution, we forced a prefetch of this library into the application's address space to analyze its functions. Fortunately, the functions in this module were unstripped and we were able to identify functions such as `CreateExpiredDialog` and `CreateNagDialog`. The reconstructed control flow for these functions is depicted in Figure 1.

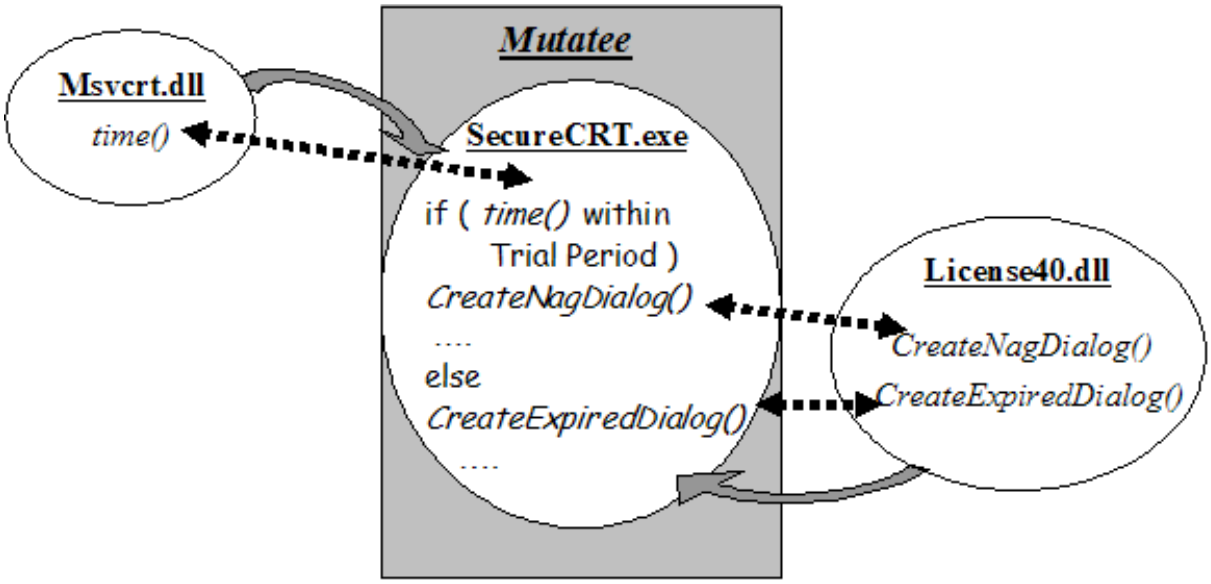

Figure 1: Normal Control Flow.

As a simplistic approach, we attempted to replace the function call to the `CreateExpiredDialog` function with a call to the `CreateNagDialog` function and were briefly thrilled to see the unexpired dialog window of the application show up in place of the expired dialog window for an overdue date. However, the application continued into the License Wizard and we were denied access to the SSH client. Returning to our attempts to analyze the program execution sequence, we extracted and compared several traces of the program for the different modules under conditions of normal operation and expiry. Our attempts to identify function calling points were hampered by the fact that the API was unable to determine the identity of the called function for several key locations including the calling points in the `main` function. Ultimately, we decided to take a different approach when we noticed calls to the `time` function in the unstripped module `msvcrt.dll`, which we knew to be the standard C library. Knowing a valid return value for this function, we replaced the function with a user-defined function that always returned a valid (unexpired) time value. After minor modifications to this function, the instrumentation worked and we were able to successfully operate the application beyond its expiry date. The structure of our mutator is depicted in Figure 2. Our instrumentation of the `time` function affects all calls made to it. In other words, all calls made to `time` will cause a branch to the function `newTime`, returning our instrumented value. We found this approach necessary since we observed

at least four distinct calls made to this function during application startup and a couple more thereafter. We subsequently tested the instrumented application extensively and were unable to notice any aberrations in its operation.

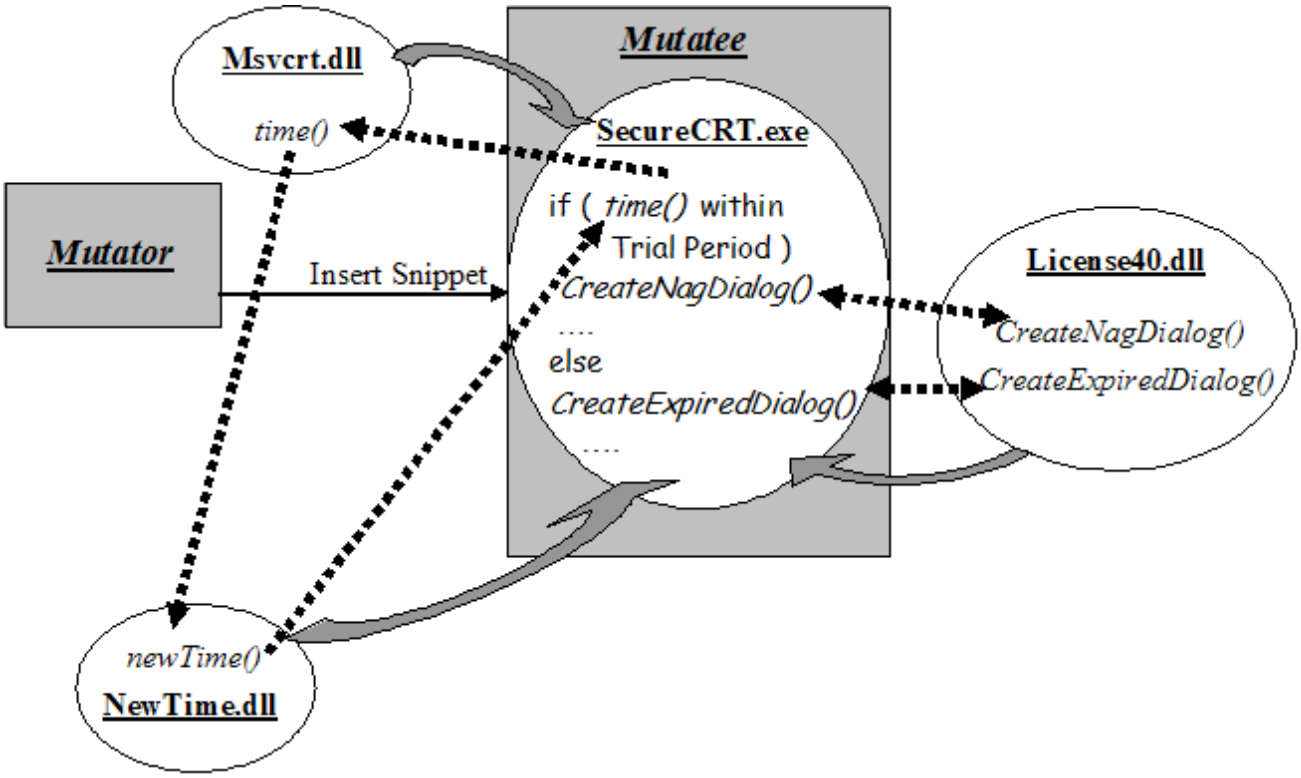

Figure 2: Control Flow after Instrumentation.

To make our mutator more generic and to avoid the extensive code analysis phase for each potential target application, we adapted our tool to extend the access of any trial executable that uses the Microsoft C runtime library, given a single valid date of operation. Our modified mutator checks for the use of `mscvrt.dll` by a specified target application, and upon such detection, replaces calls to `time` and `mktime` with a custom function that returns an unexpired time value in UTC. Since all the applications we have tested this on use a local copy of the Microsoft C library, the effect of this instrumentation is limited to the target application and does not impact other Windows programs that share the copy of this DLL in the Windows System32 directory. We have used our tool to attack other products offered on a trial basis by VanDyke Software, including VShell Server, CRT, AbsoluteFTP and EnTunnel [24], without requiring any modifications to the mutator. Out attempts to test our tool against a wider set of applications have been hampered by the fact that the DynInst API appears to have some compatibility issues with these applications. Some examples of these applications include Adobe Photoshop, Adobe PageMaker, WinZip, WinDVD and other lesser known applications such as AVISplitter, PeerGuardian, BitTorrents, SuperVideoSplitter, FreeInternetTV, Super-

VideoJoiner and AbsoluteTelnet among others. While some of these applications crashed when launched with a mutator, others threw assertion exceptions or went into infinite loops even without code instrumentation. We are still investigating this problem and hope to develop a more robust version of our tool that addresses a wider software set.

4.2 Hurdles encountered and suggested improvements

Over the course of our experience with the DynInst API in a Windows environment, we encountered various limitations with the library. As the Windows version of this library is still under development and testing, we hope our experience and feedback will prove helpful.

Issues:

- The installation of the DynInst API on a Windows system did not go as smoothly as expected. A few dependencies were missing from the original packaged release and we hope these libraries will be included in the original install bundle.

- We were not able to get Dyner [6], the interactive command-line utility based on the DynInst API to work successfully on our Windows XP host, although it has been demonstrated to work on a Windows 2000 platform. While we were able to launch the Dyner shell on our system, any attempt to create or attach to mutatee processes failed.

- We found the Windows-version of the DynInst API to be incompatible with several applications that we hoped to instrument. Some examples include applications like Adobe Photoshop, Adobe PageMaker and WinZip. While our simple test mutator crashed when attempting to load most of these applications, exceptions were thrown in the case of other applications while trying to attach to them or instrument their modules. As a result, choosing a suitable target application to instrument turned out to be a non-trivial task. We hope our feedback to the developers helps fix any residual bugs in the API and make it compatible with a wider set of Windows applications.

- One of the issues encountered with the mutator was that we were unable to instrument all the functions in the application over the course of a single execution. As a result, we had to perform incremental program tracing over multiple runs and this made it difficult to predict accurate execution sequences.

- DynInst's inability to identify the functions at several call points proved to be quite a hurdle. For instance, if we could have determined the call points for the `CreateNagDialog` and `CreateLicenseDialog` functions, we could have instrumented a simple jump from one call point to the other and displayed the appropriate dialog for an operational application.

- We were unable to determine function-specific information like parameter lists and location information for any function in our victim application.
- Some standard Windows System 32 DLLs like `shell32.dll` and `kernel32.dll` could not be instrumented. Other modules like `mfc42.dll` print junk address information and go into infinite loops when an attempt is made to instrument them.
- A welcome feature in the DynInst API would be the ability to identify all the calling points for a given function based on its name. The current implementation only serves to identify the function definition as opposed to function calls.

5 Countermeasures

While techniques for software security subversion become more ingenious and have more powerful technology at their disposal, software vendors can make their applications more robust against such attacks by following a few simple approaches. Realistically speaking however, no matter how well you protect your code, someone will be able to crack the security. Our attempt here is therefore to suggest a few measures that will make life harder for the hacker.

- **Stripped binaries**: Removing symbol table information from the executables is a simple way to avoid exposing eye-catching functions names like "CreateLicenseWizard" or "CheckLicense". Most compilers for the Windows platform do this by default, and it is an option that can be configured easily on others.
- **Stripped libraries**: In the case of the trial SecureCRT application, its calls to the unstripped Microsoft C runtime library were visible and provided an avenue of attack. To counter such an attack, we recommend removing symbol table information from all the Microsoft standard libraries or using stripped custom libraries for system calls.
- **Software (de)modularity**: While code modularity is a revered software engineering principle, it unfortunately serves to aid attacks based on binary instrumentation. Function call graphs change under different conditions of execution and this helps the attacker identify key functions of interest. An application that performs license checking within the body of the `main` function would hence be more robust to an attack as compared to one that calls a separate function to do so.
- **Multiple check points**: Performing checks on multiple occasions can also help deter an attack. For instance, rather than performing a single check at the time of invocation, an application could perform

multiple checks, either at random intervals of time during execution or upon specific user-triggered events, like saving a file.

- **Distribution of functions**: Distributing the task of license checking across multiple functions can help obfuscate the function call graph under conditions of license expiry. Such diversionary tactics in turn make it difficult to pinpoint the location of instrumentation for a successful attack.

- **Storing license data covertly**: Storing multiple copies of license key values in covert locations such as the tail end of data files or in the Windows registry, and checking for synchronicity among all these values poses another burden of detection on a potential hacker.

- **Non-trivial return values**: Rather than have a license checking function return obvious values like true and false, a more robust application could have the function return less intuitive values like for instance, a character string that includes among other information, the time of the check, or a custom data type whose value changes every run.

- **Prevent simple function replacement**: An easy measure to prevent simple replacement of a key function would be to have the function perform certain initializations or change certain conditions which could be checked for in a later section of the code. In this case, the function would be missed upon replacement and cause the application to abort.

- **Avoid obvious system calls when possible**: While this is easier said than done, applications should make an effort to avoid obvious system calls like `time` or `GetSystemTime` whenever possible. Alternatively, custom functions could be written to perform these tasks and thereby make execution sequences less traceable.

- **Avoid time-based licenses**: Rather than offer trial applications that expire after a certain period of time, applications that expire after a certain number of trials or other similar criteria provide better avenues for security.

The measures listed above cannot guarantee software protection from a determined hacker, but can certainly foil simple attack strategies involving code instrumentation. Several existing commercial products like PC Enforcer [25] and Easy Licenser [26] implement some of these practices to help deter software piracy. In addition to these products, there also exist stringent licensing schemes like the Windows Product Activation scheme [27], the Windows Digital Rights Management scheme [28] and other encryption-based product activation schemes that add another layer of software security.

6 Future Work

This effort was intended as a demonstration of the use of dynamic instrumentation technology to circumvent software license checking on a Windows platform. It is in no way intended to be a comprehensive or fool-proof attack methodology. For instance, our attack strategy could potentially be simulated by simply changing the system clock on a Windows host. This approach however has the following problems: the system time would have to be reset repeatedly to gain unlimited access to a trial application, changing the system time affects all the applications on the system and can cause aberrant behavior in the case of some applications like web browsers and finally, a modified system time can expire the licenses of other applications on the system. In contrast, our tool provides an easy and flexible way to extend the use of an application in a sandboxed fashion.

On the other hand, the availability of powerful dynamic instrumentation capabilities such as those offered by the DynInst API can be exploited to construct a more comprehensive attack toolkit. Some welcome features would be the ability to look for license files storing initial timestamps or license key values and instrument them, the ability to identify and change the return values of key functions involved in license checking and the ability to counter some of the measures described in the earlier section. Taking this approach further, a tougher requirement would be the ability to circumvent advanced licensing schemes like the Windows Product Activation scheme, Windows Digital Rights Management and other encryption-based technology.

7 Conclusions

Our experience with dynamic code instrumentation brings to light how simple tools created using libraries like DynInst can easily attack a large complement of licensed software. We hope our demonstration and our suggested list of countermeasures serves to alert members of the security community and interested software vendors to the threats of program instrumentation technology and encourages them to come up with more security measures targeted at resisting such a class of attacks. We also hope our experience and feedback with the Windows release of the DynInst API proves useful to its developers.

8 Acknowledgements

We would like to convey our thanks to Prof. Barton Miller for giving us the opportunity and the support to work on the DynInst API. Thanks are also due to Laune Harris for helping us troubleshoot the API on the Windows platform and to our referees, Vikas Garg and Kyle Rupnow for their insightful comments and

helpful feedback on our work.

References

[1] DynInstAPI Programmer's Guide

[2] Bryan Buck and Jeffrey K. Hollingsworth: "An API for Runtime Code Patching"

[3] Tevfik Kosar, Mihai Christodorescu and Rob Iverson: "Opening Pandora's Box: Using Binary Code Rewrite to Bypass License Checks"

[4] Barton P. Miller, Mihai Christodorescu, Robert Iverson, Tevfik Kosar, Alexander Mirgorodskii and Florentina Popovici: "Playing inside the Black Box: Using Dynamic Instrumentation to Create Security Holes"

[5] Laune Harris: Analysis of Stripped Binary Code

http://www.cs.wisc.edu/paradyn/PCW2004/stripped_lharris1.ppt

[6] Jeffrey K. Hollingsworth and Mehmet Altinel: "Dyner User's Guide", Release 4.1

[7] Naveen Kumar, Jonathan Misurda, Bruce R. Childers and Mary Lou Soffa: "FIST: A Framework for Instrumentation in Software Dynamic Translator" Naveen Kumar Jonathan

[8] Han Bok Lee and Benjamin G. Zorn. BIT: "A tool for instrumenting Java bytecodes. In Proceedings of the USENIX Symposium on Internet Technologies and Systems Proceedings", 1997, pages 73-82, 1997.

[9] James R. Larus and Eric Schnarr: "EEL: Machine-independent executable editing. In Proceedings of the ACM Conference on Programming Language Design and Implementation", pages 291-300, 1995.

[10] Amitabh Srivastava and Alan Eustace: "ATOM-A system for building customized program analysis tools", In Proceedings of the ACM Conference on Programming Language Design and Implementation, pages 196-205, 1994.

[11] Ted Romer, Geoff Voelker, Dennis Lee, Alec Wolman, Wayne Wong, Hank Levy, Brian Bershad and Brad Chen: "Instrumentation and Optimization of Win32/Intel Executables Using Etch"

[12] VJ Reddi, A Settle, DA Connors and RS Cohn: "PIN: A Binary Instrumentation Tool for Computer Architecture Research and Education"

[13] G. Desoli, N. Mateev, E. Duesterwald, P. Faraboschi and J. Fisher: "DELI: A new runtime control point", Int'l. Symp. on Microarchitecture (MICRO-35), November 2002.

[14] Vasanth Bala, Evelyn Duesterwald and Sanjeev Banerjia:"Dynamo: A transparent dynamic optimization system", 2000

[15] M. Arnold, S. Fink, D. Grove, M. Hind and P. Sweeney:"Adaptive optimization in the Jalapeno JVM", Conf. on Object-Oriented Programming, Systems, Languages and Applications, Oct. 2000

[16] C. Jaramillo, R. Gupta, and M. L. Soffa: "FULLDOC: A full reporting debugger for optimized code", Proc. of Static Analysis Symposium, 2000.

[17] Pin Zhou, Feng Qin, Wei Liu, Yuanyuan Zhou and Josep Torrellas: "iWatcher: Efficient Architectural Support for Software Debugging", 2004

[18] R. M. Stallman and R. H. Pesch: "Using GDB: A guide to the GNU source-level debugger", GDB version 4.0. Technical report, Free Software Foundation, Cambridge, MA, 1991.

[19] V. Kiriansky, D. Bruening and S. Amarasinghe: "Secure execution via program shepherding", USENIX Security Symposium, August 2002.

[20] K. Scott and J. Davidson: "Safe virtual execution using software dynamic translation", 2002 Annual Computer Security Applications Conference, Dec. 2002.

[21] Barton P. Miller, Mark D. Callaghan, Jonathan M. Cargille, Jeffrey K. Hollingsworth, R. Bruce Irvin, Karen L. Karavanic, Krishna Kunchithapadam and Tia Newhall: "The Paradyn Parallel Performance Measurement Tool", IEEE Computer 28, 11, (November 1995): 37-46. Special issue on performance evaluation tools for parallel and distributed computer systems.

[22] Victor C. Zandy, Barton P. Miller, and Miron Livny: "Process Hijacking", in 8th International Symposium on High Performance Distributed Computing (HPDC '99, Redondo Beach, California, August 1999): 177-184.

[23] Van Dyke Software: Secure CRT: Secure Shell Solution

http://www.vandyke.com/download/securecrt/index.html

[24] Van Dyke Software Products:

http://www.vandyke.com/products/index.html

[25] PC Enforcer:

http://www.skaro.net/enforcer/index.html

[26] Easy Licenser:

http://www.agilis-sw.com/ezlm/index.htm

[27] Windows Product Activation scheme:

http://www.yak.net/fqa/256.html

[28] Josh Cohen, Microsoft Corporation: "A General Overview of Windows Media DRM 10 Device Technologies"