Mississippi State UNIVERSITY

J. A. "Drew" Hamilton, Jr., Ph.D.

Director, Distributed Analytics & Security Institute
Director, Center for Cyber Innovation
Professor, Computer Science & Engineering
Fellow, Society for Modeling and Simulation

CCI
Post Office Box 9627
Mississippi State, MS 39762

Voice: (662) 325-2294
Fax: (662) 325-7692
hamilton@cci.msstate.edu

Simulation Security: Securing the Future of Simulation

What if a simulation is too good?

Outline

- What is simulation security and why should you care?
- A case study in simulation software vulnerability analysis.
- Simulation security state of the art.
- Conclusions and a proposed way ahead.

HACKERS CAN TURN YOUR HOME COMPUTER INTO A BOMB

. . . & blow your family to smithereens!

By RANDY JEFFRIES / *Weekly World News*

WASHINGTON — Right now, computer hackers have the ability to turn your home computer into a bomb and blow you to Kingdom Come — and they can do it anonymously from thousands of miles away!

Experts say the recent "break-ins" that paralyzed the Amazon.com, Buy.com and eBAY websites are tame compared to what will happen in the near future.

Computer expert Arnold Yabenson, president of the Washington-based consumer group National CyberCrime Prevention Foundation (NCPF), says that as far as computer crime is concerned, we've only seen the tip of the iceberg.

"The criminals who knocked out those three major online businesses are the least of our worries," Yabenson told *Weekly World News*.

"There are brilliant but unscrupulous hackers out there who have developed technologies that the average person can't even dream of. Even people who are familiar with how computers work have trouble getting their minds around the terrible things that can be done.

"It is already possible for an assassin to send someone an e-mail with an innocent-looking attachment connected to it. When the receiver downloads the attachment, the electrical current and molecular structure of the central processing unit is altered, causing it to blast apart like a large hand grenade.

"As shocking as this is, it shouldn't surprise anyone. It's just the next step in an ever-escalating progression of horrors conceived and instituted by hackers."

Yabenson points out that these dangerous sociopaths have already:

- Vandalized FBI and U. S. Army websites.
- Broken into Chinese military networks.
- Come within two digits of cracking an 87-digit Russian security code that would have sent deadly missiles hurtling toward five of America's major cities.

"As dangerous as this technology is right now, it's going to get much scarier," Yabenson said.

"Soon it will be sold to terrorists cults and fanatical religious-fringe groups.

"Instead of blowing up a single plane, these groups will be able to patch into the central computer of a large airline and blow up hundreds of planes at once.

"And worse, this e-mail bomb program will eventually find its way into the hands of anyone who wants it.

"That means anyone who has a quarrel with you, holds a grudge against you or just plain doesn't like your looks, can kill you and never be found out."

Sickos can wreak death and destruction from thousands of miles away!

Arnold Yabenson.

KABOOM! It might not look like it, but an innocent home computer like this one can be turned into a deadly weapon.

Unreasonable Security Fears

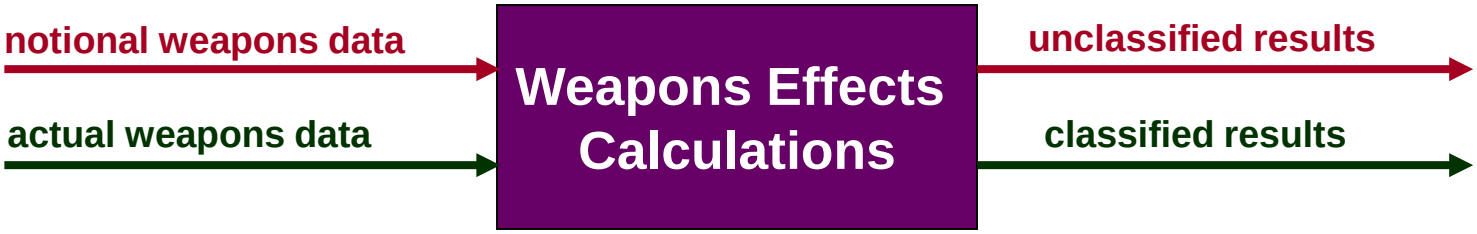
Simulation Customers in the USA

- **Department of Defense**
- **Walt Disney Corp.**

The Missile Defense Agency Asked:

- What is the impact of sharing missile defense simulation software with our Allies?
- What exactly are we sharing?
- What sensitive information can be gleaned from the internals of the software?

Typical DoD Security Model
Weapons Training

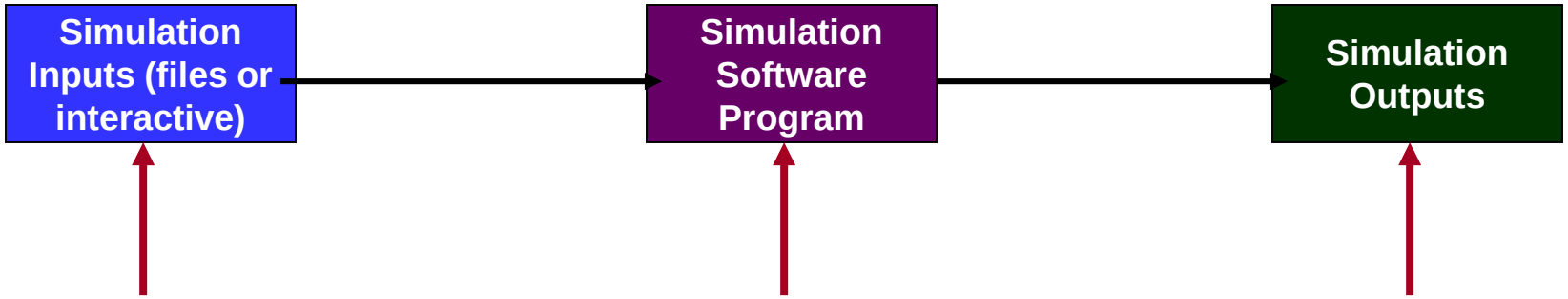


- **Assumptions**
  - **Calculating weapons effects are already well known, only the actual weapons capabilities are classified**
  - **The calculations themselves do not reveal sensitive information about training, tactics and procedures used in weapons targeting**

Is this model appropriate for missile defense simulations?

Attacking Simulations as Software

- **OBJECTIVES**
  - Look for the underlying models the simulation is constructed from
  - Compromise training, tactics and procedures used in missile defense
  - Compromise weapons and systems performance data



1. Experimentation w/"open source" system data
2. Privilege escalation via buffer overflows
3. Analysis of bounds checking if implemented

1. Exploitation of operating system vulnerabilities
2. Analysis of installed files
3. Decompilation and disassembly of targeted executables

1. Sensitivity Analysis of output based on input changes
2. One "off" test cases to examine relationships

High Assurance Vulnerability Assessment

- Line-by-Line verification of source code
- Professional and/or contract decompilation of executables
- Complete review of published documentation
- Analysis of simulation runs to evaluate training, tactics and procedures
- Open source review of weapons and systems data
- Analysis of degree of parameterization

A Software Engineering Approach to VA

Challenges in checking source code

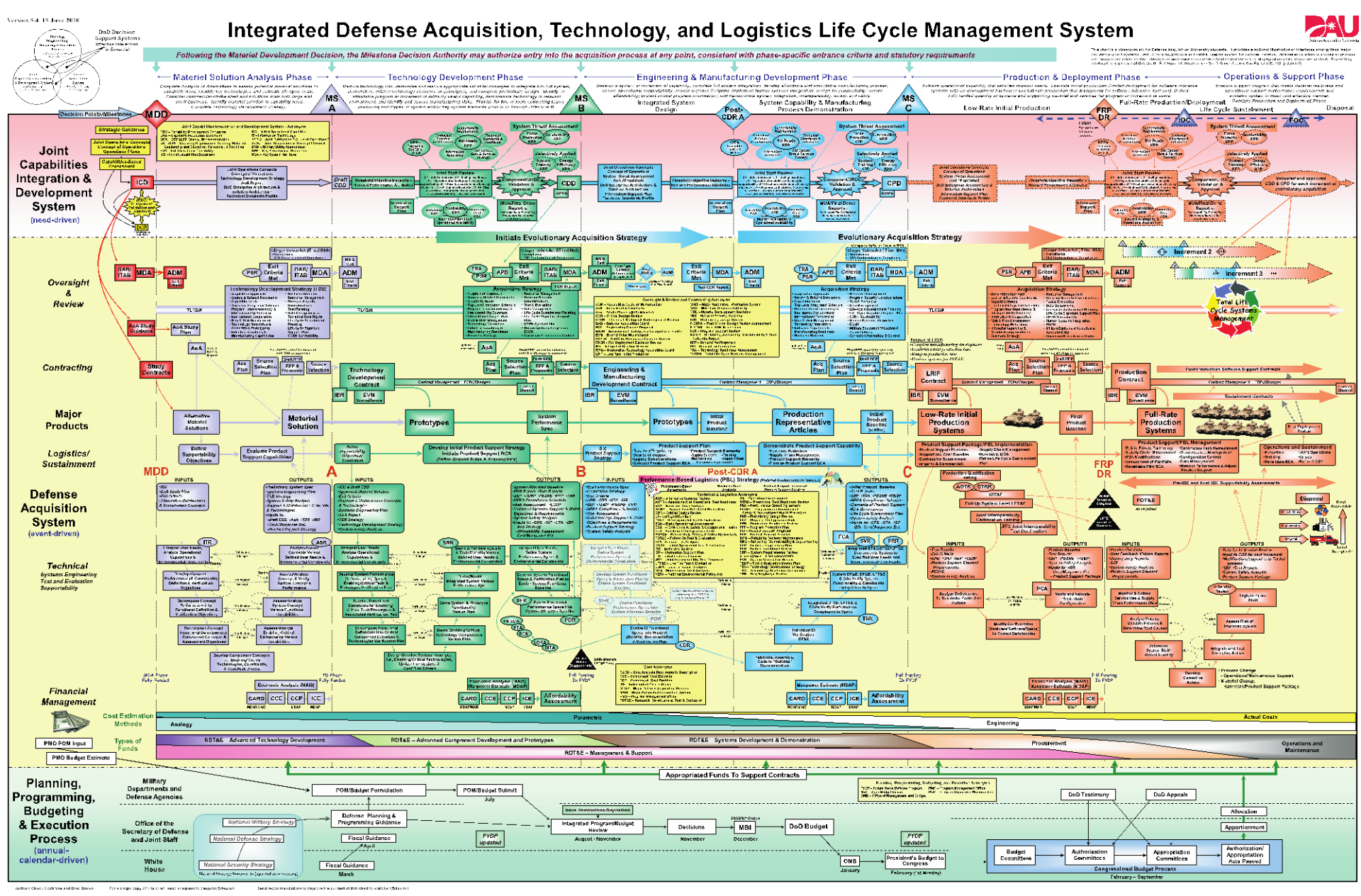
VOLUME I—PARTS 1 TO 51

FEDERAL ACQUISITION REGULATION

ISSUED MARCH 2005 BY THE:

GENERAL SERVICES ADMINISTRATION
DEPARTMENT OF DEFENSE
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

(This edition includes the consolidation of all Federal Acquisition Circulars through 2001-27)


Integrated Defense Acquisition, Technology, and Logistics Life Cycle Management System
Following the Materiel Development Decision, the Milestone Decision Authority may authorize entry into the acquisition process at any point, consistent with phase-specific entrance criteria and statutory requirements
Materiel Solution Analysis Phase
Technology Development Phase
Engineering & Manufacturing Development Phase
Production & Deployment Phase
Operations & Support Phase
MS A
MS B
MS C
MDD
FRP DR
IOC
FOC
Integrated System Design
System Capability & Manufacturing Process Demonstration
Low Rate Initial Production
Full-Rate Production/Deployment
Life Cycle Sustainment
Disposal
Joint Capabilities Integration & Development System (need-driven)
Oversight & Review
Contracting
Major Products
Logistics/ Sustainment
Defense Acquisition System (event-driven)
Technical
Financial Management
Initiate Evolutionary Acquisition Strategy
Evolutionary Acquisition Strategy
Increment 2
Increment 3
Total Life Cycle Systems Management
Post-CDR A
Performance-Based Logistics (PBL) Strategy
Planning, Programming, Budgeting & Execution Process (annual-calendar-driven)
Military Departments and Defense Agencies
Office of the Secretary of Defense and Joint Staff
White House
Appropriated Funds To Support Contracts
POM/Budget Formulation
POM/Budget Submit
Integrated Program/Budget Review
Decisions
DoD Budget
DoD Testimony
DoD Appeals
Allocation
Apportionment
Budget Committees
Authorization Committees
Appropriation Committees
Authorization/ Appropriation Acts Passed
Congressional Budget Process
February – September

Review of published documentation

- Conducted a major review of the more than one thousand pages of documentation.
- Concluded that the simulation indeed has a high degree of parameterization.
- While the physics calculations are sometimes complex, there was nothing to indicate any restricted or sensitive information.
- Documentation appears very consistent with the performance of the program.

Open source review of weapons and systems data

- **Classify results into four categories**
  - **Located: values were located in open sources**
  - **Derived: values derived from known values or other derived values**
  - **Guessed: parameters for which researcher input an arbitrary but seemingly reasonable value**
    - **Important to note that researcher was a computer scientist, not a subject matter expert**
  - **Default: default values in GUI used in simulation run**

Security by Obscurity

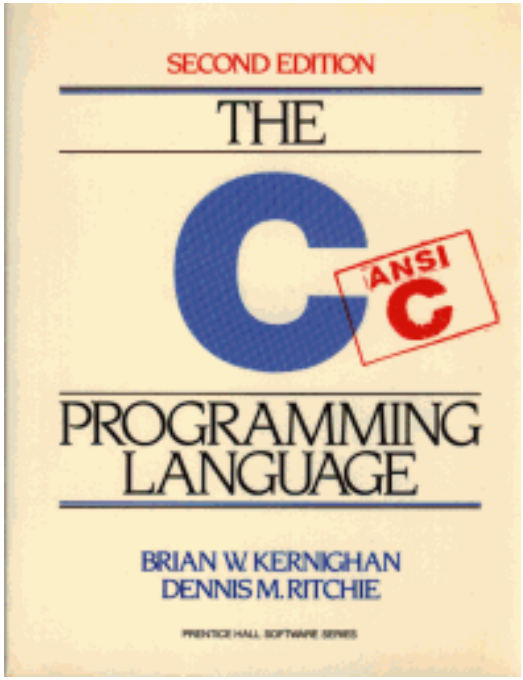
- Hide the source code and only release the executable.
- False belief that code compiled into binary remains secret just because the source is not available.
  - Java byte code is particularly vulnerable
- Netscape POP (post office protocol) 1999
  - password with weak cryptography
  - stored in windows registry
  - experimentation with XOR on password strings
    - pattern detected
    - encryption algorithm reverse engineered

Ken Thompson

- Lead developer of UNIX in early 1970's
- Installed back door that automatically added his account and password to every UNIX system
- Back door was not in the source code it was hidden in the binary code that was needed to build UNIX
- Back door automatically propagated itself into future UNIX distributions
- Revealed 14 years later in his ACM Turing Awards acceptance speech
- http://www.acm.org/classics/sep95

C, an average programming language

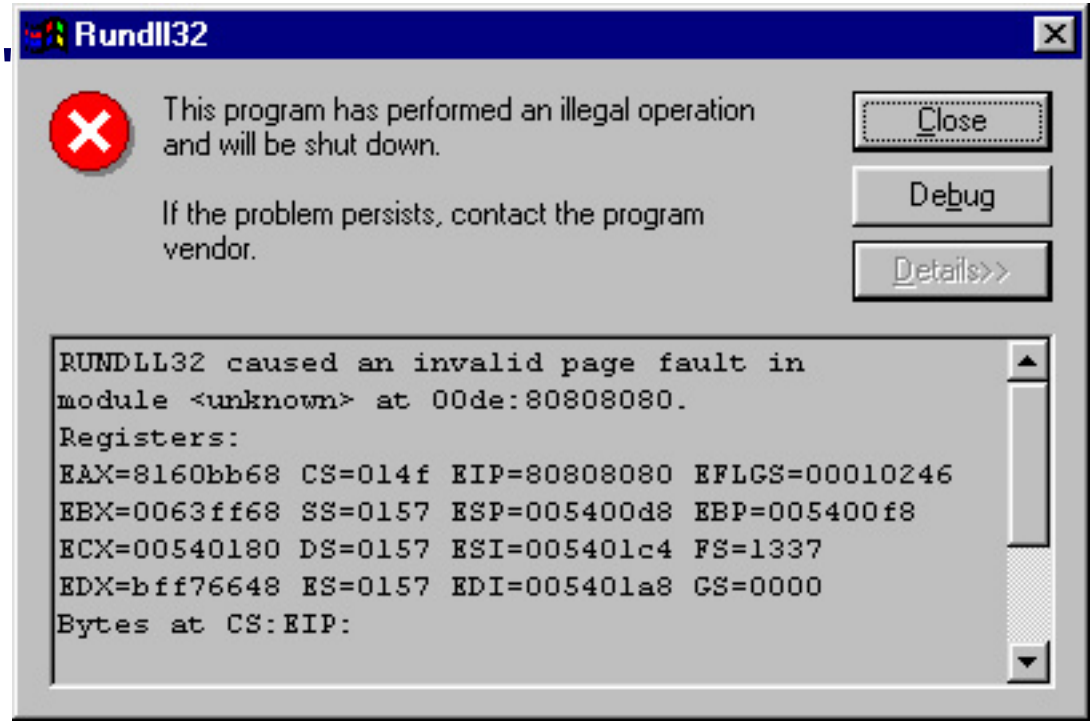
- C is inherently unsafe – programs may overflow buffers at will.
- No runtime checks that prevent writing past the end of a buffer.
- Reading or writing past the end of a buffer can cause a number of diverse behaviors
  - Programs may act in strange ways
  - Programs may fail completely
  - Programs may proceed without any noticeable difference in execution.



Notes from the Cult of the Dead Cow

- To get this to happen, I fed a string of 0x80 bytes into a popular conference package called 'Microsoft Netmeeting' through the address field of a 'speeddial' shortcut.
- EIP happens to be 0x80808080.
  - Guess what?
  - That's good!
  - I found a stack overflow!
- Now all I have to do is craft my exploit string to have some fun code inside, and tweak four of those 0x80 bytes to point to my exploit string.

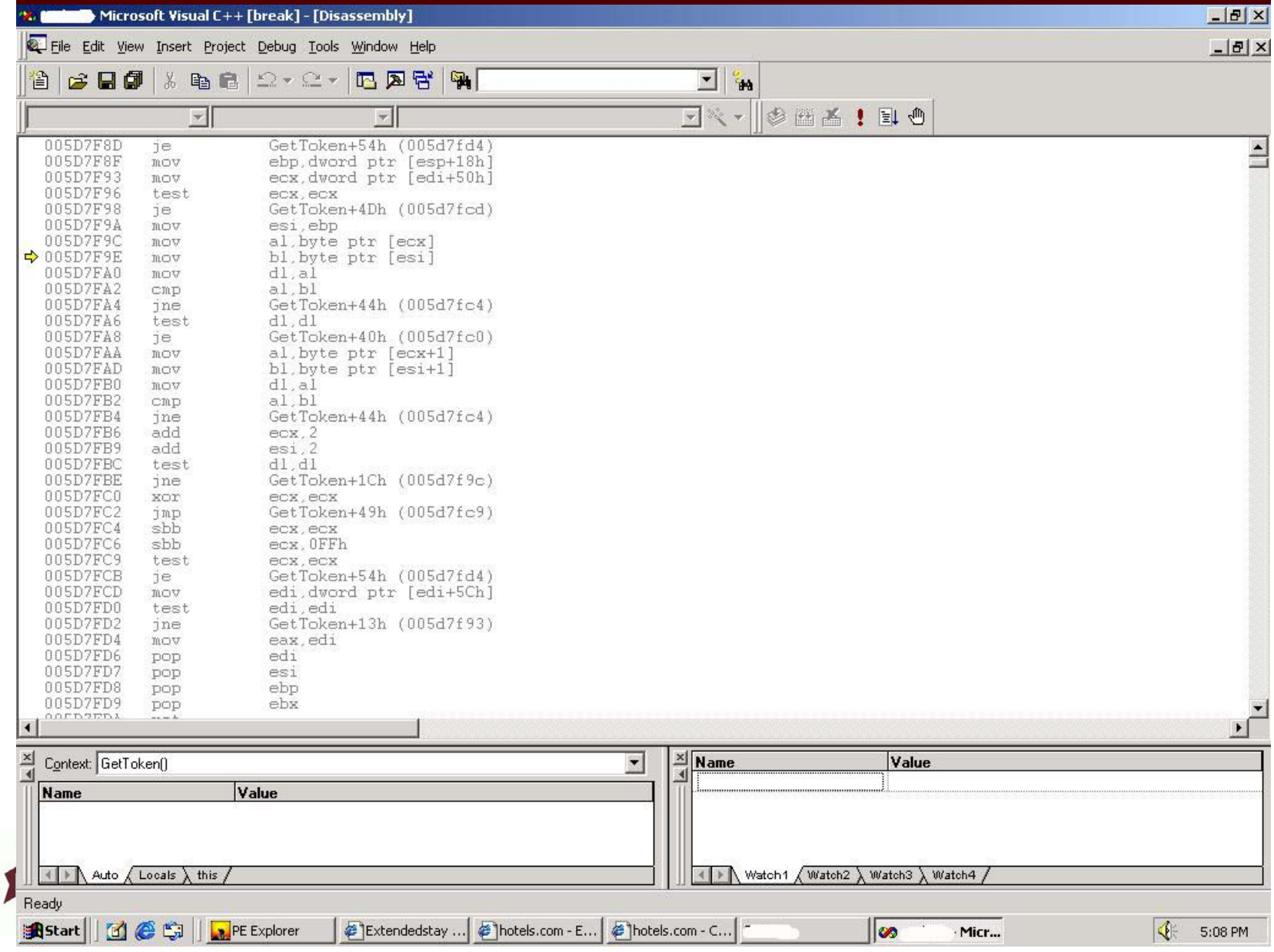
buffer overflow tutorial



http://www.cultdeadcow.com/cDc_files/cDc-351

Warning: Foul language on this site

Finding our own buffer overflow



Reverse Engineering

- Machine code analysis
- Core Dumps
- Reverse Engineering Tools
- Debuggers usually have disassemblers
  - takes machine code and translates into assembly language
  - C code versus assembly
    - loops versus counters and jumps
- Decompilers are not as mature as dissassemblers
  - attempt to convert machine language into high-level language constructs
  - JVM programs much easier to reconstruct than “hand coded” assembly language
  - decompilation performance can be enhanced if the program is compiled with debugging options on
- Assume that binary code can be reconstructed

Copy Protection

- Tradeoff – protection of intellectual property versus hassling legitimate users
  - OPNET Example
- License Keys – A psychological deterrent
  - Encryption Keys
    - use 36 character set less “1” “l” “0” “O” = 32 characters
    - Use CBC and say “Blowfish” and produce valid keys
    - Each key is a counter concatenated with a fixed binary string, encrypted and converted base 32
  - Checking the license key for validity
    - decode the base 32 string, decrypt the binary with the stored encryption key and to see that the last 12 bytes are equal to our stored binary string
- Force software to run off of distribution CDs
- Theoretically, no media is “copy proof.”

Code Obfuscation

- **Anti-tampering**
  - **Checksums**
  - **Check for debuggers**
    - **Running debuggers reset the instruction cache on every operation**
    - **Check for this condition and jump your code to crash the program**
- **Obfuscation**
  - **Rename all variables in code to arbitrary names**
  - **Automated code obfuscation still an open research area**
  - **JVM retains much more data than other HLLs**
  - **Makes programs harder to maintain**

Obfuscation Techniques

- Add code that never executes or that does nothing
  - Make calculations more complex
- Move code around
  - Spread related functions as far apart as possible
  - Fake “encapsulation”
  - Combine multiple unrelated functions into a single function
- Encode your code oddly
  - Picking strings directly out of memory is easy
  - Convert strings to odd character sets, only make strings printable when necessary
- Encrypt program parts
  - Generally “low grade” because of performance considerations
  - Data versus operation encryption
  - Hex editor for manual encryption
  - Encryption of padding

Desk check of selected source code

- Source code for some simulation modules shipped to US users.
- Desk Check done in two parts:
  - First finding security vulnerabilities that might allow a third party to take control of the simulation executable.
    - No unbounded buffers located
    - Tab key buffer overflow found earlier
  - Second search sensitive information contained in the simulation source code.
    - Conditional and assignment statements searched
    - Keyword search
- Results submitted to MDA
  - Working in a classified environment, unclear how sensitive our findings were.

Disassembly of executables

Executable	Executable size	Assembly size
Simulation	25792k	131624k
Kernel Interface	39440k	141016k
Simulation GUI	37504k	167528k

- 440,168k of assembly code was generated from three executable files totaling 102,736k in size.
- This volume of generated assembly code represents approximately 9.3 million lines of assembly code.
- Disassembled code was successfully reassembled and executed.
- 400 MHz Intel Celeron processor with 128 MB of RAM.

Analyzing the disassembled code

- The simulation assembly code totaled about 500MB in size, and was therefore difficult to work with.
- With sufficient resources, the large amount of assembly code could be understood and mapped out.
  - Also, given the number of viable decompilers that are targeted at specific compilation platforms, along with available theory on how to attempt such focused efforts [Housel 1974, Breuer 1994, Weide 1995], it would be possible for an organization to implement their own custom decompiler specifically tasked with compromising a single executable.
  - Things that might thwart such an effort would be the use of optimizing (or other obfuscating) compilers.
  - A failed disassembly attempt using PE Explorer did reveal a compiler version number "6.0." This led to successfully guessing the compiler used MS Visual C++ 6.0.

Stripping the executables

- At the end of the analysis of the compiled executables, it was discovered that neither the Windows nor the Solaris versions of the simulation had been stripped of debugging information.
- This is particularly disturbing given the information that could potentially be obtained simply by running the application through a debugger.
- For example, by running the GUI program through Microsoft's Visual C++ 6.0 Debugger, the names of several different functions could be found.
  - In addition to the function names, the number and type of arguments required by the function were also found.
  - This could greatly assist anyone seeking to compromise the simulation code, even if they did not have access to anything other than the compiled executables.

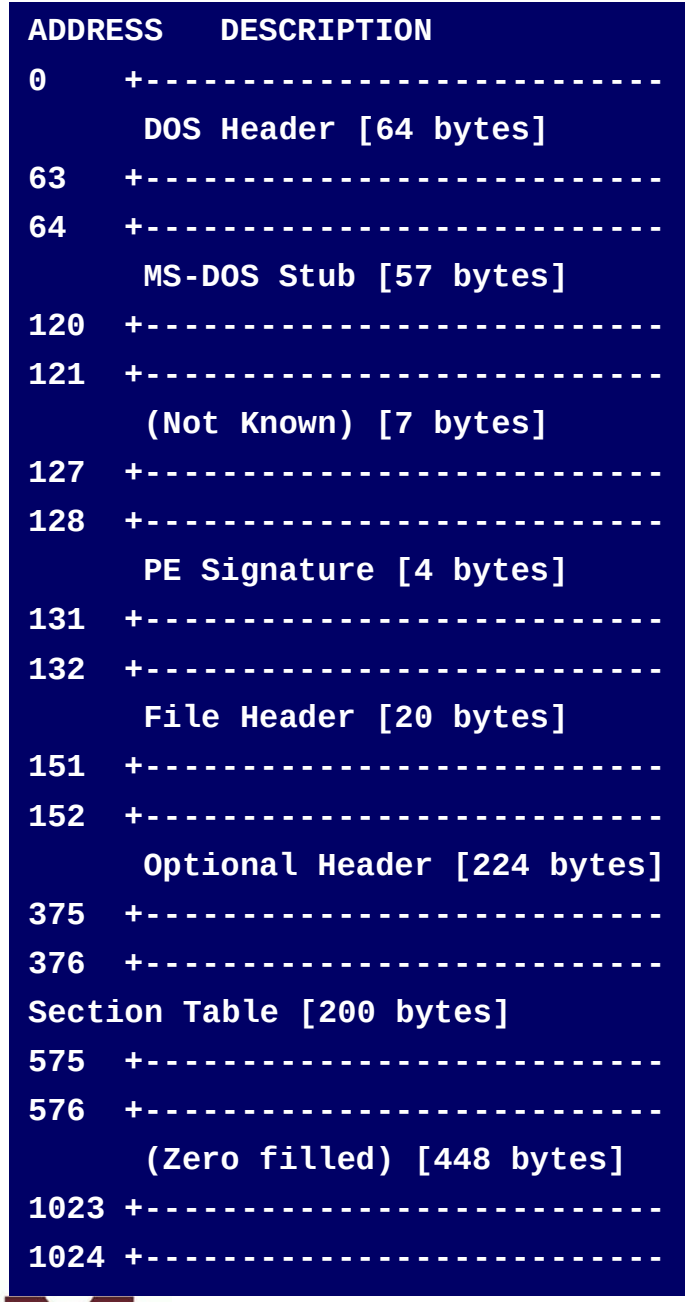
Decompilation/Analysis of Binaries

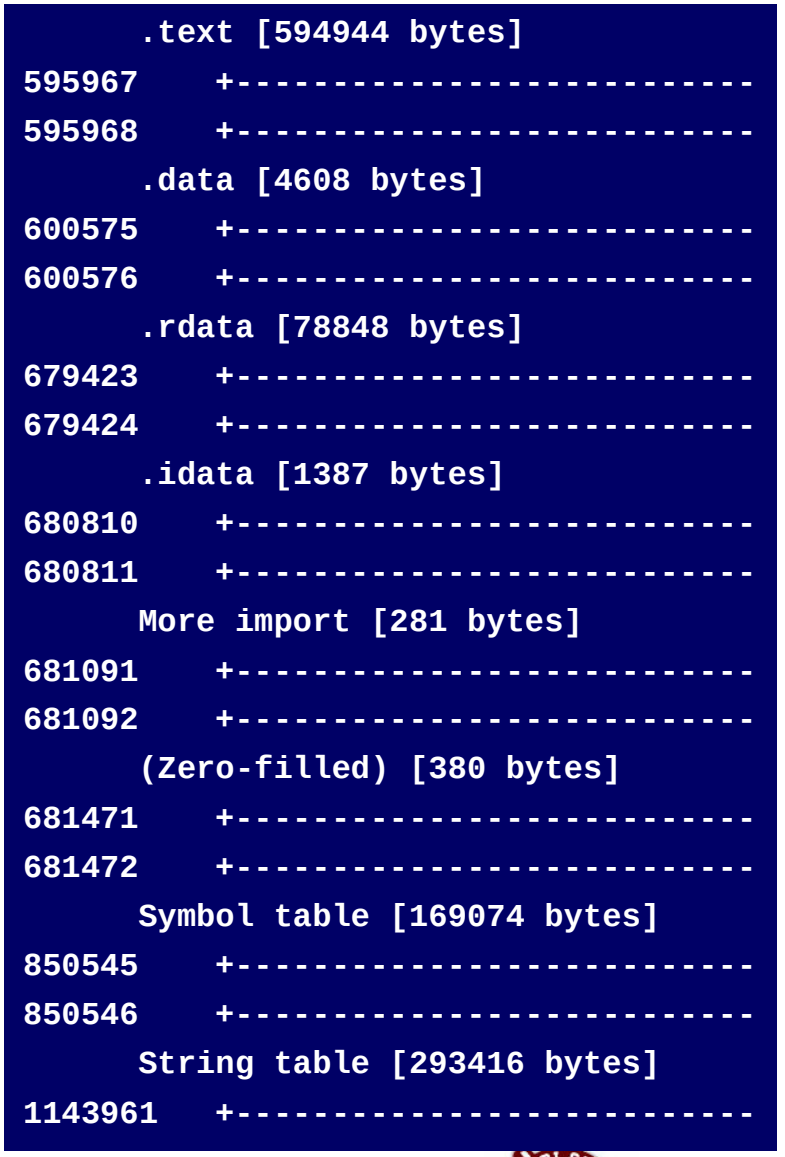
- Approximately 27 megabytes of string literals were extracted from the three executables.
- Just under 1.6 million individually discernable strings greater than or equal to four characters in length were generated.
  - Note that a large number of these strings are "trash" strings having no English-language meaning, or are object-file specific strings which have only partial English-language meaning, and which are used in computing the offsets of individual data members in certain aggregate data types.

Analysis of the Binaries

- Of the slightly less than 1.6 million strings literals found above, less than one-third, or about 450,000, of these were found in the initialized data space of the executable.
  - This included what appeared to be function names and variable/member names.
- Between 32,000 and 36,000 of these string literals appeared to be format strings of the type used in standard I/O print statements.
  - Error statements such as "Error, cannot open file %s for reading." or "MAJOR ERROR!!! [System/Ruleset/Sensor/Com Device/Jammer] %s does not exist for opfac %s," to other informational statements such as, "The following Platforms have the '%s' system type:" or "The following Systems use the '%s' system as a weapon …"
  - No weapon-specific string literals were found in this manner, except a few instances of "PATRIOT" located in an error messages such as, "Missile type %s not found in PATRIOT missile preference table."
  - No references were found which contained the strings "SCUD," "THAAD," or "Aegis."

PE File Format





Analysis of Simulation PE Files

- **findSSV tool by Dr. Jay Tevis**

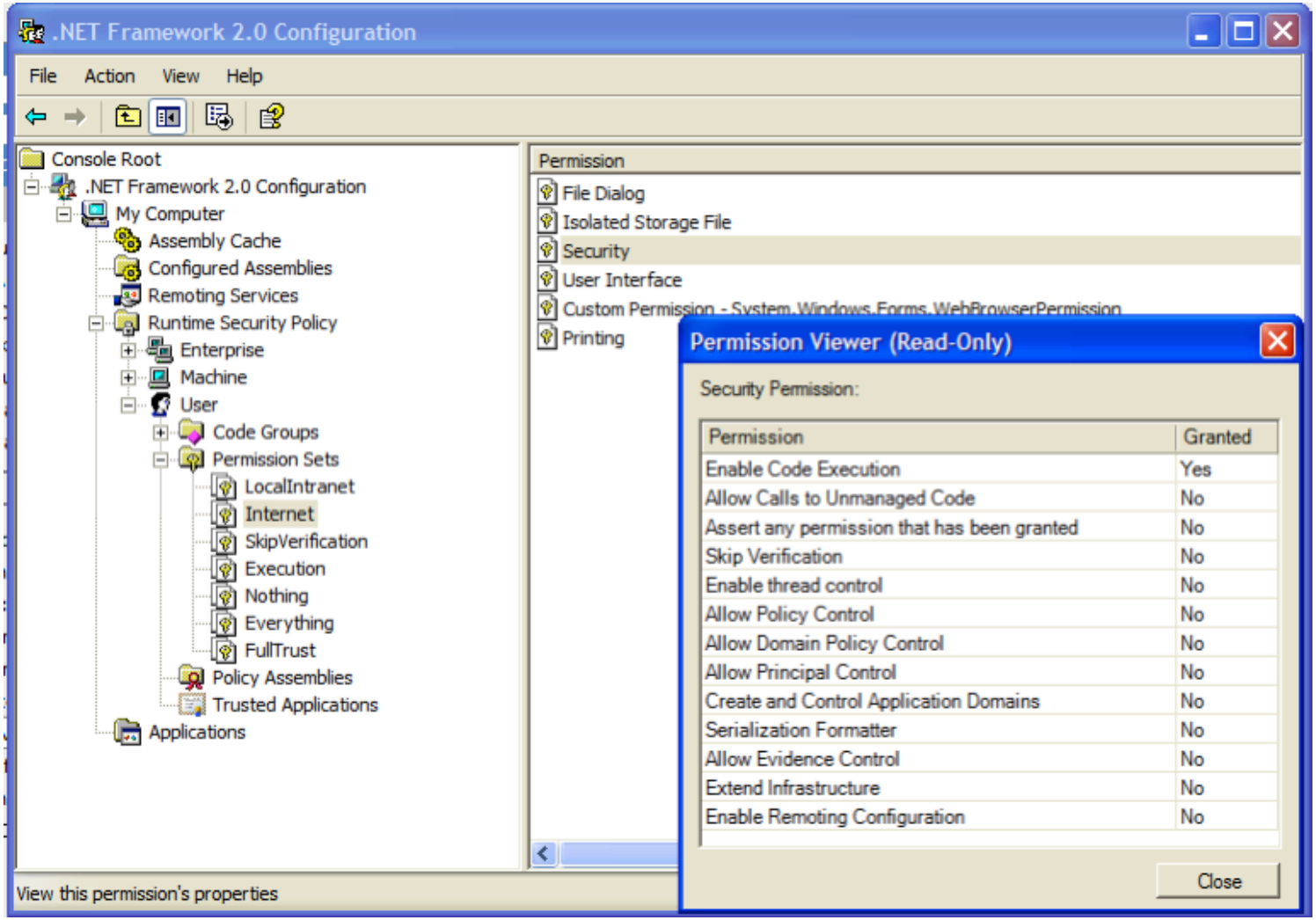
File Nbr	File Size (bytes)	Total Vul.	Large Unknown Region (bytes)	Unused Zero-filled Bytes	Import Table Anomaly	Symbol and String Tables	Debug Table
1	6,622,124	12	4,381,612	100,726	80/1620	No	yes
2	4,961,816	12	3,356,184	56,560	80/1620	No	yes
3	34,304	0	0	0	40/927	No	no
4	4,841,964	13	3,269,100	76,212	80/1694	No	yes
5	34,816	0	0	0	40/927	No	no
6	23,255,612	14	16,046,652	314,600	180/4663	No	yes
7	23,043,168	12	15,219,658	364,746	100/2495	Yes	yes
8	26,864,140	14	19,544,588	413,280	160/8341	No	yes
9	27,627,392	14	19,791,744	443,924	160/5339	No	yes
10	6,041,004	12	4,124,076	72,142	80/1556	No	yes
11	942,138	0	0	0	60/2533	No	yes
12	31,232	0	0	0	40/898	No	no
13	4,207,940	13	2,856,260	68,720	80/1570	No	yes
14	7,696,928	12	5,083,680	98,792	80/1623	No	yes
15	33,280	0	0	0	40/908	No	no
16	16,384	1	0	0	80/432	No	no
17	4,385,052	13	2,951,452	64,918	80/1594	No	yes
18	374,436	10	280.228	5022	40/1026	No	yes

We look for:
1) sections in a file whose contents can be both written to and also executed,
2) large unused zero-filled regions in a file, and
3) the use of functions susceptible to buffer overflow attacks.

Results of the Case Study

1. The simulation is vulnerable to buffer overflow attacks
2. Large number of string literals in compiled executables, particularly in Solaris version
3. Neither Windows nor Solaris version of the simulation had the debugging information stripped from the executables at compile time.
4. There is potentially sensitive information still in the source code distributed to all US customers of the simulation.
   - These instances were usually found in comments still placed in the code, usually detailing upgrades that were made to the software.
   - There were a few instances of values being hard-coded into the source code.
   - Most of the values used by the simulation appear to be input from another location, such as a file or the keyboard, or declared in header files that were not included with the simulation.

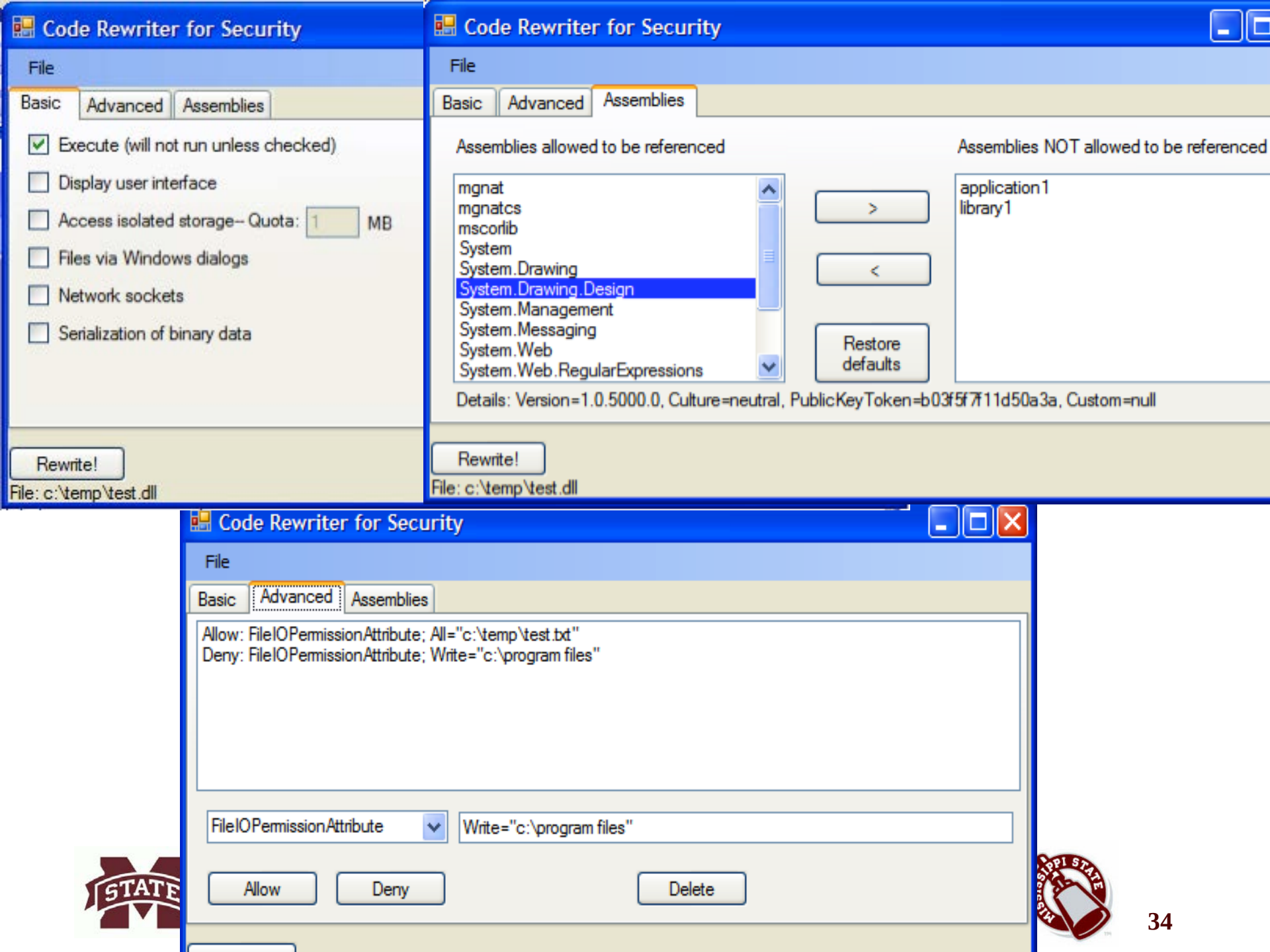
Security at a Higher Level of Abstraction



Microsoft .NET Security Configuration Tool

Concept of a .NET Assembly Rewriter

- The proposed code rewriter will use a combination of adding declarative security and rewriting byte code to ensure that an untrusted module can be safely redistributed.
- The code rewriter will use Microsoft's ILDASM disassembler to get a disassembled text file.
- It then will parse this file and create a second assembly text file containing the modifications.
- Finally the modified assembly is reassembled using Microsoft's ILASM assembler.


Code Rewriter for Security
File
Basic
Advanced
Assemblies
Execute (will not run unless checked)
Display user interface
Access isolated storage-- Quota:
1
MB
Files via Windows dialogs
Network sockets
Serialization of binary data
Rewrite!
File: c:\temp\test.dll
Code Rewriter for Security
File
Basic
Advanced
Assemblies
Assemblies allowed to be referenced
Assemblies NOT allowed to be referenced
mgnat
mgnatcs
mscorlib
System
System.Drawing
System.Drawing.Design
System.Management
System.Messaging
System.Web
System.Web.RegularExpressions
>
<
Restore defaults
application1
library1
Details: Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a, Custom=null
Rewrite!
File: c:\temp\test.dll
Code Rewriter for Security
File
Basic
Advanced
Assemblies
Allow: FileIOPermissionAttribute; All="c:\temp\test.txt"
Deny: FileIOPermissionAttribute; Write="c:\program files"
FileIOPermissionAttribute
Write="c:\program files"
Allow
Deny
Delete

Software Security at Higher Level of Abstraction

- The use of type-safe managed code or virtual machines allows for useful security guarantees
  - (in particular, the absence of buffer overflow errors).
  - Although .NET provides a significant security framework, its focus is on individual administrators protecting their machines or networks from untrusted code, not on allowing developers to include untrusted modules in new software projects.
  - Furthermore, the framework is overly complex, which means it is unlikely to be used correctly.
- We propose a tool which allows developers to rewrite .NET assemblies so that they can be redistributed with security guarantees that are enforced by the .NET framework.
  - This tool will have a very simple interface and is sufficiently flexible to create any possible security policy.
  - The code rewriter also provides the ability to choose simple security policies that should cover most cases.

Simulation Software Security Summary

- Best defense on buffer overflows is implicit bounds checking.
- Machine language executables cannot be considered inherently secure.
  - Source code not required to compromise compiled software.
- Executable software once released cannot be controlled.
- Training, tactics and procedures embedded in a compiled software simulation are vulnerable to compromise if released.
- Reverse engineering techniques have limitations
  - Reverse engineering by resource unconstrained professional intelligence efforts can over time make significant discoveries.
- The future of simulation software security is working at a higher level of abstraction.

Questions?

What do you want to talk about?