Lecture 14

System Integrity Services
Obfuscation

OS independent integrity checking

- Observation
  - Majority of critical server vulnerabilities are memory based
  - Modern anti-virus software must scan memory
  - Modern malware must disable, tamper with, or circumvent such scanning
    - Bagle and Lion worms attempt to kill anti-virus scanning
- Intel's solution: SMM/AMT
  - System Management Mode starring the Active Management Technology platform
  - Use isolated trusted firmware solution to ensure the presence and integrity of anti-virus software

Problems with current IDS

- Host IDS
  - Rootkit allows unrestricted system access to enable attacker to thwart checks
  - Example: Witty
- Network IDS
  - Insertion, evasion attacks
- VMM (virtual machine isolation)
  - IDS running in VMM vulnerable to attack from DMA device
- Programmable DMA device to scan host memory (Petroni)
  - Difficult to do since page tables not located at well-known addresses

Threats

- Image modification
  - Modify in-memory executable code of agent
    - Permanent or transient (restore when integrity check runs)
- Disable agent
  - Kill agent, steal interrupt vectors of agent, modify OS scheduler to bypass agent
  - IDSes that do memory read verification only (and not execution verification) are vulnerable
- Out-of-context jump
  - Trick IDS into signing bogus “health reports” by jumping directly to its signing code
- Dynamic data modification
  - Modify important data agent uses (i.e. filtering rules) or program state
- Interrupt hijacking
  - Modify state of agent when it is interrupted asynchronously

Threats

- System Management RAM cache attack
  - Modify cached copy of SMM handler to execute malicious code in SMM
- Multi-processor attack
  - Run code in parallel to agent on MP systems to cause undesired effects
- Exploit software defects
  - Buffer-overflow, format string attacks
- Agent circumvention
  - Bypass checking code by reimplementing code without check in it (i.e. filters in device drivers)
- DMA attack
  - For VMM systems, instruct an I/O device to perform a DMA directly into VMM to “inject” into “isolated” environment
- Theft of secrets
  - Steal secrets used by agent (keys) to impersonate them

Verification approach

- Locality
  - Location in memory that program resides
  - Do not allow requests from outside of agent's memory to gain access
- Integrity
  - Check whether or not an agent has been modified from its original state
- Execution state
  - Check whether agent is running or being scheduled to run over time
- Problems
  - Does not address tampering with dynamic data
  - Function pointers?

SIS

- Host agent initialization
  - SIS specific initialization code section executed when agent is loaded
  - Registers with IMM (integrity management manager)
    - Locations of critical sections in host virtual memory
    - Location of its integrity manifest
- Integrity manifest
  - Signed summary description of critical components of host software agent
    - Section table
    - Relocation table
    - Symbol table
    - Mirrored from ELF and PE formats
  - Signed by private key of host software agent manufacturer

SIS

- Integrity verification
  - First check done at registration when agent begins
  - Periodically after that
  - Integrity check value (ICV) or a MAC signed by secret key stored in SMRAM
  - Problem: virtual memory implemented by OS
    - VMRS: virtual memory reconstitution service implemented
  - Subsequent checks use SMI to initiate ICV
- Locality verification
  - ISM checks CR3, CS and EIP
  - Virtual address of instruction triggering SMI calculated
  - Program identified based on previously registration with IMM
- Key manangement
  - Platform master key used to derive session keys

SIS

- ISM operation
  - Must ensure consistent measurement by disallowing modifications when measuring
  - Details in paper
- Virtual memory reconstitution
  - Maps host virtual address to physical address while running on isolated partition
- Dynamic data protection
  - Perform integrity checks to detect changes to protected dynamic data
  - Only reasonable to do on small amount of very critical data

Agent execution detection

- Use IMM to generate ICV over a heartbeat message
  - Send signature to host agent
  - Host agent appends to heartbeat message sends it out
  - IMM double-checks message to ensure integrity of heartbeat

Threats revisited

- Image modification
  - Integrity checks detect this
- Disable agent
  - Detected since authentic heartbeats stop
- Out-of-context jump
  - Partially handled using locality (SMI tracing)
- Dynamic data modification
  - Signing and verifying source and validity of dynamic data updates
- Interrupt hijacking
  - Disabling interrupts and verifying it within SMI handler

Threats revisited

- System Management RAM cache attack
  - SMI flushes cache upon exit
- Multi-processor attack
  - Stop other cores and threads upon running SIS
- Exploit software defects
  - Addressed by No-EX bit?
- Agent circumvention
  - Tie critical functions to SMI
- DMA attack
  - SMRAM inaccessible to DMA due to hardware restrictions
- Theft of secrets
  - Stored in SMRAM

Open problems

- Kernel data structure integrity
  - Interrupt Vector Table
- DLL integrity
- Integration with platform security tools
  - Firewalls, IDS

Obfuscation

Objectives

- Slow reverse engineering process
  - Make automated analysis difficult
  - Make code more complicated
  - Make decompilation difficult
  - Make code unreadable by human

Metrics

- Resilience
  - Irreversibility
- Cost
  - Added run-time or code size
- Stealth
  - Similarity to rest of code

Techniques

- Data obfuscation
- Control flow obfuscation
- Advanced techniques

Data obfuscation

- Renaming variables, procedures, classes, methods
- Deleting comments and spaces
- Inserting dead code
- Variable splitting
- Scalar/object conversion
- Change variable lifetime
- Split/fold/merge arrays
- Change encoding
- Merge scalar variables

Control-flow obfuscation

- Break basic blocks
- Inline methods
- Outline statements
- Unroll loops
- Reorder statements
- Reorder loops
- Merge all functions into one

Advanced techniques

- Reuse identifiers
- Misleading comments
- Modify inheritance relations
- Convert static data to procedural data
- Store part of program as text and interpret it only during runtime
- Remove library calls
- Attack specific decompilers and debuggers

Shiva (Mehta/Clowes 2003)

- Outer encryption layer
  - Defeats "strings"
  - Slows access to protected code
- TRAP flag detection
  - Defeat single-stepping
- "checkme" data check
- ptrace defense
  - Exits if ptrace active
  - Clones itself and two copies ptrace each other to prevent additional PTRACE_ATTACH "inter-ptrace"
- Timing checks
- AES, password protected middle encryption layer
- Inner encryption layer
  - Run-time protection

Shiva (Mehta/Clowes 2003)

- /proc defenses
  - Only portions of binary decrypted at a given time
- INT 3 instruction replacement
  - Some instructions replaced with INT 3
  - Instructions emulated in INT 3 handler
  - If debugger uses INT 3, code will be missing
- Jumping into middle of instrucitons
- Polymorphic code generation

Reversing Shiva

- Use similar techniques to run partially and dump images
  - Scripted decryption via IDA scripts
  - Virtual x86 plugin for IDA

.NET reversing

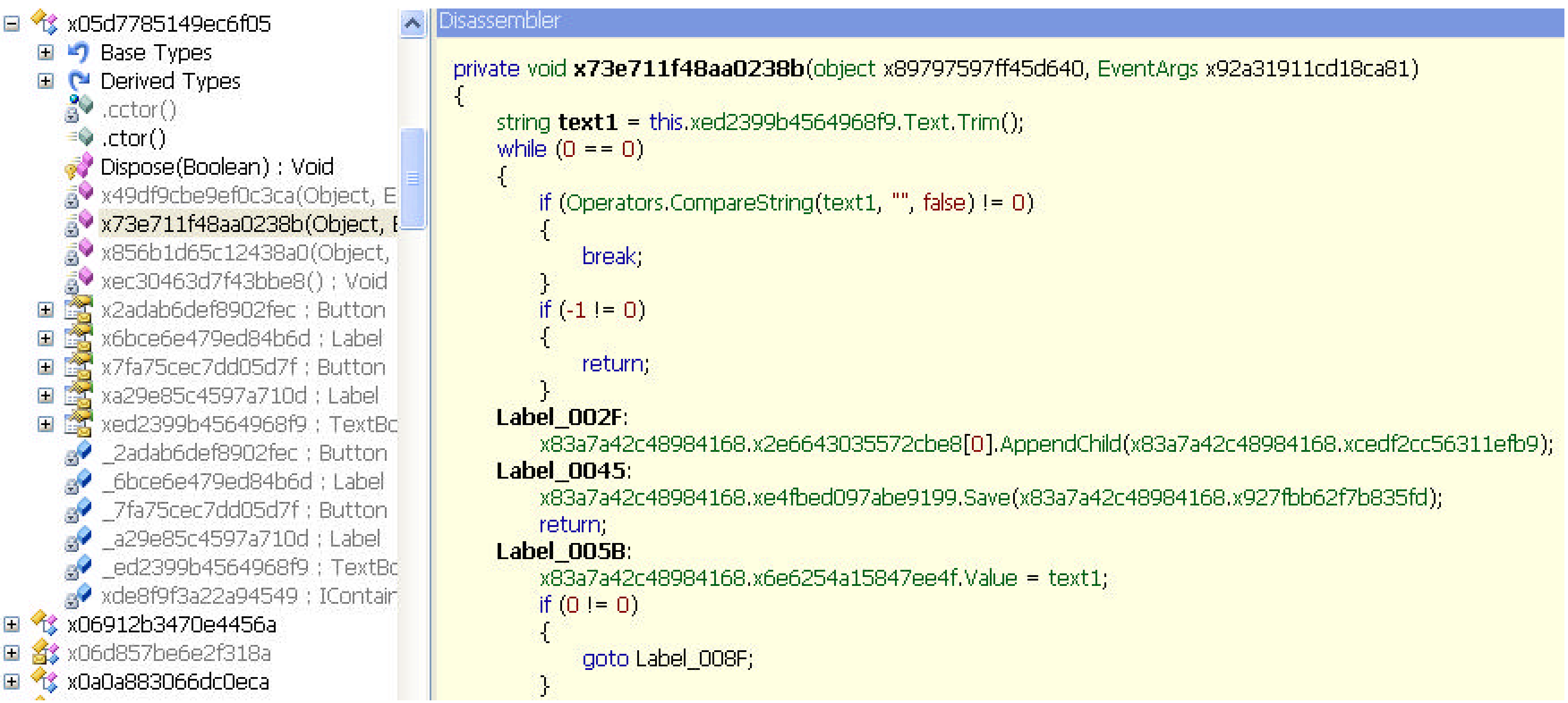
- Reversing tutorial on .NET
  - http://accessroot.com
  - http://www.blong.com/Conferences/DCon2003/ReverseEngineering/ReverseEngineering.htm
- Tools
  - ILDASM
    - Disassembler that comes with .NET framework SDK
  - Reflector
  - Dis#

.NET obfuscation

- http://www.codebreakers-journal.com/index.php?option=com_content&task=view&id=123&Itemid=97
- StrongName
  - Verifies code integrity via cryptographic hash calculation
  - Prevents patching
  - Easily bypassed via ildasm edits to remove signature scheme
    - http://www.andreabertolotto.net/
    - StrongName Remove
  - Or patch system DLL to make check return valid all the time

.NET obfuscation

- Name obfuscation
  - Change metadata saved with binary to make names either unprintable or random



.NET obfuscation

- Flow obfuscation
  - Make msil reading hard by preventing its translation into a HLL
  - Adding boolean checks that are always true or false
  - Splitting source into many segments and connecting them using various branches
  - Mess with stack
    - MSIL is stack-based and will not allow unballanced stack
    - Insert a “pop” that will never run
    - Breaks Reflector

```
.assembly extern mscorlib { }
.assembly extern System{}
.assembly sample {}
.method public hidebysig void Main()
{
.entrypoint
br.s start_here
pop
start_here:
ldstr "hello!"
call void [mscorlib]System.Console::WriteLine(string)
ret
}
```

.NET obfuscation

- Metadata encryption
  - String references stored as metadata in managed PE file
  - Often the key in reversing
  - Encrypt to hide and decrypt just before use

Dotfuscator

- http://www.preemptive.com/products/dotfuscator/FAQ.html
- Uses a variety of mechanisms to obfuscate
- All done after compilation (i.e. does not modify source code)

Dotfuscator

- http://www.preemptive.com/products/dotfuscator/FAQ.html
- “Overload Induction" renaming
  - Identify colliding sets of methods across inheritance hierarchies
    - Rename such sets according to some enumeration (e.g. the alphabet or unprintable characters).
  - Method overloading is induced on a grand scale
    - OI algorithm determines all opportunities for name reuse and takes advantage of them.
    - Can use return type to determine method uniqueness as well
  - Anecdotal evidence
    - 33% of ALL methods were renamed to a single character (such as "a").
    - Typically, 10% more are renamed to "b", etc.
      - overload induction reduces the final program size of obfuscated code.
  - Up to 10% of the size savings in Dotfuscated and DashO'd programs

Dotfuscator

- Undoing Dotfuscator renaming
  - Decompiler needs to implement overload induction themselves (ironically, violating Preemptive's patent in the process) to undo it.
  - Overload induction is provably irreversible
    - The best reversing will come out with a different number of unique methods than the original source code contained
    - Overload induction destroys original overloading relationships
    - In reversed state, there will be no overloaded methods.
      - Grand designers of OO technology implemented overloaded methods as a way of creating "more readable code“
      - By removing that ability, the code has less information in it than before.

Dotfuscator

- Also supports
  - String encryption
  - Incremental obfuscation (for patches)
  - Control-flow obfuscation
    - Breaks loops and other HLL control structures up