Outline

• Purposes
• Code obfuscation
  – control-flow graph obfuscation
  – variable hiding
• Digital signatures
• Code randomization
• Virtual machines
• Automated Theorem Proving
Security and Compilers

• Tamper resistance
  – Code should run as shipped
  – Binary executables should not be modified

• Resistance against attacks
  – Buffer overflows
  – DoS attacks

• User security
  – Running untrusted binaries

• Attack scenarios
  – Attacker has access to
    • binaries, source code, a debugger, a service
Code Obfuscation

- Code obfuscation makes the (compiled binary) code difficult or impossible to understand
- Different kinds of obfuscation
  - Flow-graph obfuscation
  - Variable hiding
Flow-Graph Obfuscation

• What are these control-flow graphs doing?
The Big Switch

- Any control flow-graph can be turned into a giant switch statement contained in an infinite loop
  - Each block becomes a separate case
  - A variable controls which block to go to next
  - The cases can be arbitrarily numbered
Variable Hiding

• We would like to obfuscate the values of variables in our program
  – Transform variables using algebraic (or bitwise boolean) transformations

• Example:
  – Replace x with u and use the u=x+27262
  – 'if (x > 10)' becomes 'if (u > 27262)'

• Example:
  – Replace y with v and use v=y*4635
  – 'if (y > 10)' becomes 'if (v > 46350)'
  – y+7 becomes v + 32445
  – x > y becomes 4635*(u - 27262) > v
Variable Hiding

• Variable Hiding can be arbitrarily complicated
  – Can use cryptographic hash functions
    • if \((x == 10)\) becomes:
      
      \[
      \text{if (hash(x) == 38934782782928)}
      \]

• Hiding the variable that controls flow is especially confounding
  – Transform \(x\) using addition, multiplication, and/or bitwise operations
  – \(\text{switch (hash(x)) \{ case 83434727: ... case 2382722: ... \}}\)
Obfuscating Compilers

• Several companies make obfuscating compilers
  – Angel Security
    • Randomized obfuscating compiler
    • Receives millions in funding from US DoD
  – Cloakware
    • Builds DRM around obfuscating compilers
    • iTunes, Blu-Ray, PVRs
Digital Signatures

- Public/private digital signature schemes exist
  - Private key allows owner to sign any binary object
  - Public key allows anyone to verify a signature

- Use digital signature schemes to check that binaries were not tampered with
  - Software developer signs their binary
  - Code in the software checks the signature on the binary
  - Modifying the binary invalidates the signature and this can't be fixed without the public key
Digital Signatures

• Problem:
  – Code in the binary checks the digital signature:
    • if (!verify_signature(this_file)) { exit(-1); }
  – Attacker can just delete this code
  – Signature is invalid, but no one checks it

• Solutions:
  – Have the OS check the signature
  – Spread the signature checking throughout the binary
    • More work for attacker
  – Obfuscate the signature checking
  – Obfuscate variables using a value that depends on a hash of the binary
Randomizing Compilers

- Compilers have a lot of choice they can make
  - Order of functions in a binary
  - Order of variables on the stack
  - Order of global variables on the heap
  - Order of instructions within a basic block
  - Locations of basic blocks (especially with obfuscating)

- Randomizing compilers make some or/all of these choices randomly
Randomizing compilers

• Security advantage
  – More difficult to perform a buffer overflow attack if you don't know the layout of the binary

• IP rights enforcement advantage
  – Vendors can ship a different binary to every customer
  – Can identify customers who violate copyright
Virtual Machines

- Some languages compile for use on virtual machines
  - JVM is an example

- A secure VM can enforce security policies
  - No file access
  - No network access
  - No access to personal information
  - No sharing of personal information (through data-flow analysis)
Automated Theorem Proving

• Some VMs prove things about the binaries before executing them
  – the value of x does not affect the value of y
  – the value of x does not affect any value transmitted over the network
  – ...

• The VM may refuse to run some code if it can not prove the theorem it needs
  – Problem: Reasoning about code leads to undecidable problems
Proof-Carrying Code

• To help with automated theorem proving, some code is *proof carrying*
  – Finding proofs of theorem takes a long time
  – Checking proofs is easy

• The code comes with proofs of statements like
  – the value of x does not affect the value of y
  – the value of x does not affect any value transmitted over the network
  – ...

• The VM can load the program much faster this way
Summary

• Code obfuscation
  – Makes code difficult to
    • understand, reverse-engineer, or modify (while still preserving correctness)

• Digital signatures
  – Make it difficult to modify code

• Code randomization
  – Can make every executable unique
    • Acts as a watermark
    • Makes some attacks more difficult

• Virtual Machines and Theorem Proving
  – Allows fine-grained control over what a is allowed to do