Objective

• Today’s question – given secure cryptographic primitives, how can we build secure software implementations?

Software Security

Outline

• Motivation
• Obfuscation
• Tamper Resistant Software
• Software Attestation

Software Threat model

• Category I
  – Malicious threat outside of PC
  – “Black-Box”
• Category II
  – Malicious attack originates as software running on the platform
  – Viruses, Trojans
• Category III
  – Adversary has complete control of the platform (system owner)

Digital Rights Management

• Content protection
  – Market examples: set-top boxes, digital TV, iTunes, WM-DRM

Cat III

• The adversary model
  – Full access to the software implementation
  – Full access to the execution host
  – CPU calls, memory pages, cache, OS libraries
  – All the tools that the adversary uses, share similar privileges
  – Can run the program at any time, from any entry point.

• The Malicious Host Model (Software Agents)
  [Sander and Tschudin, 1998]
• The White-Box Attack Context (WBAC)
  [Chow et al., 2002]
Digital Rights Management

• Breaking the content protection
  – Isolation of decryption routine or content key
  – Tamper the REM
  – Circumvent the authentication

Digital Rights Management

• DRM technologies: AACS (Advanced Access Content System – Blu-ray and HD DVD), CSS (Content Scrambling System – DVD), WM-DRM, PVP (Protected Video Path), Adobe Acrobat DRM, FairPlay (iTunes)

... appear to be broken over and over again: DeCSS, AnyDVD

• Systems themselves (and the cryptographic primitives they employ) are not broken, but the implementation thereof appears to introduce vulnerabilities
  – The weakest link in the chain implementation of cryptographic primitives
  – Desperate desire for strong software protection techniques

Attacks

• Software Attacks
  – Debuggers, emulators, reverse engineering tools, ...
  – Entropy Attack
    • Keys need to be chosen at random from a (uniform) distribution \( \rightarrow \) high entropy.
    • Code typically has low entropy

(Sander and Tschudi, 1998)

Key Whitening Attack

• Implementation attack on ciphers that deploy a key-whitening (e.g., AES)

\[ y = S(x) + k \]

• Strategy: identify and overwrite S-box definition in binary:
  
  \[ S \rightarrow 0, \text{then } y = 0 + k \]

(Kerins and Kursawe, 2006)

Software Security

• Software security (in this presentation): implement applications such that they are resistant against
  – Leakage of information
    • Code: IP protection
    • Data: cryptographic information, valuable info
  – Illegitimate modification (tampering)
    • To ensure correct execution
    • Liability

... and many others (but not covered in this presentation)
  – Discourage modification or misbehaviour (watermarking, fingerprinting, traitor tracing)
  – Penetration testing
  – Proof carrying-code
  – Secure programming (against buffer overflows, bugs, correctness)
  – NOT: malicious software (malware/viruses) – Cat II

Outline

• Motivation
• Obfuscation
• Tamper Resistant Software
• Software Attestation
Code Obfuscation

• Protection of software against unintended information leakage (confidentiality): code obfuscation
  – Garble the internals of a program in such a way that they become difficult to read/understand

secret

“I don’t know what this does”

IOCCC

• The International Obfuscated C Code Contest (IOCCC)
  - http://www.ioccc.org
  - http://www.crackmes.de

IOCCC

main()
{
    printf("Hello World!");
}

2 Major Analysis Classes

• Main classes of analysis techniques
  - Static Analysis
    • Program comprehension without executing the code
    • Code reverse engineering
  - Dynamic Analysis
    • Use execution information
    • Introduction of faults/code modifications
Code Obfuscation

• A few techniques
  – CFG flattening
  – Opaque predicates
    • Mainly against static analysis
    • Against dynamic analysis: force significant computational effort for adversary and trigger false negatives

Theory on Obfuscation

• Pragmatic approach:
  – Theoretic foundations of Obfuscation, based on theoretic models for NIZK.
    • Comparison between ‘obfuscated’ world and ‘ideal’ world

Theory on Obfuscation

• Pragmatic approach:
  – Theoretic foundations of Obfuscation, based on theoretic models for NIZK.
    • Comparison between ‘obfuscated’ world and ‘ideal’ world

Obfuscating Password Functions

• Positive results
  – Obfuscation of Point Functions
    boolean isValidPassword (String password) {
      if (password = "ADMIN")
        return true;
      else
        return false;
    }

• Positive results
  – Obfuscation of Point Functions
    boolean isValidPassword (String password) {
      if (SHA3(password) = "3d451a9ef8sf7…9f01aa4d")
        return true;
      else
        return false;
    }

Obfuscation

• Theoretic modeling
  – Barak et al. – impossibility result
  – Lynn et al., Wee – obfuscation of point functions

• Question:
  – Are there other ‘useful’ families of programs that can be obfuscated?
  – Could we obfuscate “an encryption scheme”?
    • What does this mean? → implement in a secure way.
    • But: obfuscation is against leakage of information of the program; while the security of a cryptographic scheme is defined against the program and its purpose.

White-Box Cryptography

• Aim: to implement cryptographic primitives in a secure way.
  – E.g., obfuscate AES in such a way that it remains CPA secure.
White-Box Cryptography

- **Main Idea**
  - Spread key information over a network of lookup tables
  - Make every building block seemingly independent of the key

- **Objective:** force an adversary to analyze the complete network in order to obtain secret key information \( \Rightarrow \) force to resort to black-box attacks

White-Box Cryptography

- **Memory dump:**

  ![Memory Dump](image)

  - Suggests the presence of key information

- **Main analysis techniques**
  - Differential cryptanalysis (White-Box DES)
  - Algebraic cryptanalysis (White-Box AES)
    - Mainly using algebraic equivalence solvers
    - Break PR-CPA security

WBC – Theory

- **Obfuscation with respect to a security notion**
  - E.g., AES w.r.t. KR-CPA

- **Negative result**
  - For any scheme (e.g., AES), there exists a security notion that is satisfied in ‘black-box’, which cannot be satisfied in ‘white-box’.
  - Problems with composition

- **Positive result**
  - There exists a scheme + security notion that can be obfuscated (reduction proof to a hard problem)

Conclusion

- **Obfuscation & White-Box Cryptography:** confidentiality of program information
  - Solutions have been presented
  - How to build ‘secure’ applications with these building blocks?

```java
boolean isValidPassword(password) {
    if (SHA3(password) = 3d451a9ef8sf7...9f01aa4d)
        return true;
    else
        return false;
}
```

Outline

- **Motivation**
- **Obfuscation**
- **Tamper Resistant Software**
- **Software Assurance**

Software Tamper Resistance

- **Objective:** to make software resistant to illegitimate modification

  - Early proposal – Integrity Verification Kernels: extra code, injected into the original implementation.
    - Interleaving tasks
    - Distribute secrets
    - “decrypt” memory cells
    - Run-time verification
      - Digital signatures
      - Accumulator function: \( h_i = H(h_{i-1}, m_i) \)

[Aucsmith, 1996]
Self-checking code

- Insert code that verifies the integrity of (binary) code at execution time.
  - Adversary: find each "tester", and eliminate it
- [Horne et al., 2001]: nested self-checking code
  - Testers check each other
  - Tester-code needs to be very simple (efficiency and stealthiness)

Software Tamper Resistance

- Other protection techniques
  - Double execution of code
  - Invariants monitoring
- Practical example: Skype (v. 2006)
  - Almost everything is obfuscated (looks random)
  - Anti-debugging and anti-dumping tricks
  - Encrypted' binary
  - Nearly 300 integrity checks
  - ... but broken

Cloning Attack

- Defeating self-checksumming techniques on Von Neumann architecture platforms

Outline

- Motivation
- Obfuscation
- Tamper Resistant Software
- Software Assurance

Software Assurance

- Problem statement: how to be able to guarantee the execution of software
- Definition (DHS)
  - Trustworthiness – No vulnerabilities or exploits (maliciously or unintentionally)
  - Predictable Execution – Confidence that software, when executed, functions as intended
  - Conformance – Software processes and products conform to requirements and standards
- Large interest (e.g., US task force on this)
  https://buildsecurityin.us-cert.gov/
Genuinity tests

- Question: is my software running, and is it running on the machine I think it is running?
  - Detect if adversary is running the application in a simulated (virtualized) environment
- Genuinity tests: show that software and hardware is real.
  - Realness test
  - Tests that
    - Leverage detailed knowledge about CPUs
    - Are slow to simulate
    - Potential good discriminants: TLBs, caches, miss counters, branch predications

[Kernel, 2003]

Pioneer

- Pioneer: Remote attestation by timed verification of checksum computed on a random walk through the memory
  - Guarantee of the integrity of an application at execution time
- Issues: network delay, deterministic execution time, time optimal implementation.

[Seehadri et al., 2005]

Remote Attestation

- Improvements on Pioneer have been presented, to address some issues
  - SWATT: send verification agents instead of a challenge to a pre-cooked agent
- Other approaches:
  - Trusted Computing
  - Pure software-based

TC Attestation

- TPM + CRMT

Issue: bootstrapping is required $\Rightarrow$ constraints on OS
(attempts to address this via virtualization)

Software-based Attestation

- Software-based remote SW attestation on legacy operating systems: bundling many techniques to obtain a reasonable degree of trustworthiness.

Reality Check

- An actual example: large interest from the gaming industry
  - World of Warcraft: the warden

Reality Check

- An actual example: large interest from the gaming industry
  - World of Warcraft: the warden
Software-based Attestation ++

- Software-based remote SW attestation on legacy operating system, enhanced with lightweight hardware (e.g., Smart Card or TPM)

TPM-based Pioneer

- TPM-assisted remote SW attestation
  - Address the timing issue (due to network delay) that Pioneer has: execution time of the verification is timed via the trusted clock ticker of a TPM

Reality Check

- USB Dongle
  - USB acts as a trusted server
  - At load time: keys are injected in the (obfuscated) code, which is then deployed on the untrusted machine

Conclusion

- Software Protection (from one point of view):
  - Confidentiality
    - Code Obfuscation
      - Hide functionality
    - White-Box Cryptography
      - Secure implementations of cryptographic primitives
  - Integrity
    - Software Tamper Resistance (local)
    - Problem: defeat of integrity verification
    - Remote Attestation
      - Towards a trustworthiness guarantee of software execution

A Few References

- D. Aucsmith “Tamper Resistant Software: An Implementation”, In Information Hiding 1996
- B. Barak et al. “On the (Im)possibility of Obfuscating Programs”, In CRYPTO 2001
- D. Schellekens et al. “Remote attestation on legacy operating systems with trusted platform modules”, In REM 2007
- H. Wee “On Obfuscating Point Functions”, In STOC 2005

Thank you.

https://homes.esat.kuleuven.be/~bwyseur