The Industrial Challenges in Software Security and Protection

Yuan Xiang Gu
Co-Founder of Cloakware
Senior Technology Advisor, Irdeto
Guest Professor, Northwest University

The 9th International Summer School on Information Security and Protection
Canberra, Australia, July 9 - 13, 2018
Myself Briefing

- 1975 -1988: Professor of Northwest University in China
- 1988 -1990: Visiting professor of McGill University, Canada
- 1990 -1997: Senior scientist and architect at Nortel
  - 1993: Effective Immune Software (EIS, early Cloakware idea)
- 1997 - 2007: Co-founder and executive positions of Cloakware
  - leading security research and collaboration with universities worldwide
- 2011 - present: Guest professor of Northwest University, China
- 2018.May - present: Senior Technology Advisor, Irdeto
The 1st ISSISP was held in Beijing, China, in 2009
- Jack Davidson, Christian Collberg, Roberto Giacobazzi, Yuan Gu, etc.

Have been holding in following
- 3 times in Asian (China, India)
- 3 times in Europe (Belgium, Italy, France)
- 1 time in North America (USA)
- 1 time in South America (Brazil)
- 1 time in Australia

ISSISP2019 is considering to hold in China to celebrate the 10th year of anniversary
SSPREW History

- The 1\textsuperscript{st} international workshop on Software Security and Protection (SSP) with IEEE ISI was held in Beijing, China, in 2010
  - Christian Collberg, Jack Davidson, Roberto Giacobazzi, Yuan Gu, etc.

- Since 2016, SSP has merged with Program Protection and Reverse Engineering Workshop (PPREW) into SSPREW (Software Security, Protection and Reverse Engineering Workshop) co-located with ACSAC.

- SSPREW 2018 with ACSAC is to hold in to on Descemer 3–7, 2018, Puerto Rico, USA (CFP soon)
SECURING DIGITAL ASSETS FOR 21 YEARS

50 MILLION TRANSACTIONS PROTECTED PER DAY

OVER 70 SOFTWARE-BASED CLOAKED CA CUSTOMERS WORLDWIDE WITH OVER 25 MILLION DEPLOYED CLIENT DEVICES

70 MILLION PERSONALIZED SEMICONDUCTOR CHIPS PROVISIONED VIA IRDETO'S KEYS & CREDENTIALS SOLUTION

MORE THAN 191 MILLION CRYPTOGRAPHIC KEYS GENERATED AND UNDER MANAGEMENT
Agenda

- **Part 1: Trends in Threats and Security Paint Points**
- **Part 2: New Challenges and White-box Security**
  - New Challenges to Information Security
  - White-Box Attacks in Real World
  - Software Security: More Than Vulnerability
  - Power of Software Protection
  - Software Security Immunity
  - Connected Application central based Security Model
  - AI and ML Security Problems
  - Software Security Lifecycle and Digital Asset Protection
  - New View of Information Security
- **Part 3: White-box Security Patterns**
  - Introduction to WB Computing Security Patterns
  - Description in Details of Selected WB Security Pattern
Part 1:

Trends in Threats and Security Pain Points

Days of hacking games and movies are over…

... Attacking businesses is the new trend!
SW Protection Business Trends

- Applications
- Games
- Digital Contents (TV, Video, Music, Film)
- Apps & Web Applications
- Mobile Payments
- E-Commerce
- AI & ML SW
- IOT Devices and Systems
- Connected & Intelligent & Autonomous Vehicles

Timeline:
- 1995+
- 2010+
- 2015+
Any device or sensor with an IP address connected to a system network via Internet is an entry point for hackers and cybercriminals: *Just like leaving your front door wide open for thieves*.
Cybersecurity Attacks in Connected and Intelligent Vehicles

2010
Wifi laptop remote control

2010
OnStar remote control

2015
Dealer malware propagation

2015
Key fob replicator

2015
Jeep hack

2015
BMW unlock

2015
OnStar unlock / start takeover

2015
Tesla WebKit hack

2015
Corvette insurance dongle

2016
Nissan Leaf mobile app

2016
Tesla Wifi and Android App

2017
Tesla hacked by Keen again

Automotive Cybersecurity History
2018 Cybersecurity Attacks in Connected and Intelligent Vehicles

- **January**
  - Rare Malware Targeting Uber’s Android App Uncovered.
  - Canadian Train Company Targeted by North Korean Cyberattack.
  - Australian Car-Sharing Company in Identity Theft Hack.
  - Charging Electric Cars: A Free Ride for Hackers.

- **February**
  - Thieves Hack Into Keyless Entry Fob and Steal Cars in UK.
  - Tesla Hackers Hijacked Amazon Cloud Account to Mine Cryptocurrency.
  - For Cadillac, CAN Bus data exposed by On-Board Diagnostics-sniffing.

- **March (bad month for self-driving vehicles)**
  - Failed Tesla Autopilot system caused a killing of a woman
  - Uber’s Volvo SUV killed a pedestrian woman.

- **May**
  - The team from Tencent’s Keen Security Lab discovered 14 vulnerabilities in over tens of millions of BMW connected vehicles.
Hackers Everywhere and Hacking is Big Business

- **For Fun**
  - UnSophisticated AttacKers

- **For Profit**
  - Cheaters
  - Black Business
  - Organized Crime
  - TerroRist Organizations Hacktivists

- **For Special Interests**
  - Competitors
  - Nation States
  - TerroRist Organizations

- **For Challenge**
  - Sophisticated Researchers
Cybercrime has evolved from single hackers into resilient highly skilled organizations performing global cyber attacks

- Cyber breaches recorded by businesses have almost doubled in five years, from 68 per business in 2012 to 130 per business in 2017.
- A 2017 study of 254 companies across seven countries put the annual cost of responding to cyberattacks at £11.7 million per company, a year-on-year increase of 27.4%.
- The financial impact of cybersecurity breaches is rising, and some of the largest costs in 2017 related to ransomware attacks. The cost of cybercrime to businesses over the next five years is expected to be US$8 trillion.
- Another growing trend is the use of cyberattacks to target critical infrastructure and strategic industrial sectors, raising fears that, in a worst-case scenario, attackers could trigger a breakdown in the systems that keep societies functioning.

(Source: Marsh Global Risks Report 2018)
Growing Impact Scope of Security Threats

- Millions of persons
- Billions of devices
- Millions of networks
- Millions of companies
- Millions of organizations
- Multiple nations
- Cross continents
- Any connected environments
- Any connected devices
- Anyone & Any where & Any time

- Kill and hurt people’s life
- Disorder societies
- Destroy and hurt businesses
- Damage nations and the world

- Financial lost
- Business lost
- Reduce Productivities
- Inconvenience

- Individual persons
- Individual devices
- Individual companies
- Individual networks
- Individual organizations

- Financial lost
- Business lost
- Reduce Productivities
- Inconvenience

- Kill and hurt people’s life
- Disorder societies
- Destroy and hurt businesses
- Damage nations and the world

- Multiple nations
- Cross continents
- Any connected environments
- Any connected devices
- Anyone & Any where & Any time
KrebsOnSecurity.com was knocked offline by 620Gbps DDoS. One of the biggest ever recorded. This was followed by a 1Tbps attack against French web host OVH.

Indications are that an estimated 500k+ IoT devices such as security cameras and DVRs were used as a botnet for the attack.

Botnet of refrigerators? Cars? Traffic Lights? Medical Devices?
Would we even know it was happening?
Mobile ransomware quadrupled in 2015
Fast becoming a mature, million dollar business for organized crime
35 known ransomware “products” in operation in 2015
Targeting corporations and public entities such as municipal gov’ts and hospitals
Ransomware in Healthcare

USA top target for ransomware with 320,000+ infected systems

Healthcare providers pay USD $6B annually to ransomware

Cerber “ransomware-as-a-service” takes 40% of extorted profits; run by Russian crime ring
Global Ransomware – WannaCry & Petya

- On May 12, 2017: WannaCry attacks to 300,000 machines in 150 countries worldwide

- On June 27, 2017: Petya attacks in Europe, the Middle East and the US
Software makes data, functionalities, properties, assets …

Software makes networks, connections, and communications …

Software makes applications, systems, devices, servers …

Software plays music, videos, games …

Software drives satellites, planes and connected vehicles …

Software is doing everything in digital …
Problem: All Software Is Hackable

• Unsecured software is as readable as a book – IP and critical algorithms are simple for hackers to access and exploit
• Advances in debugging and reverse engineering techniques have empowered increasingly more capable and tech-savvy hackers
• Open source software and hacker collaboration compound the problem, providing “easy learning”
Software Threats and Hackings

- Man-In-the-Middle Attacks
- Dynamic Attacks
- Static Attacks
- Man-At-the-End Attacks
- Remote Attacks
Connate Software Security Problems

Vulnerability + Lack Immunity = Software is so easy to attack
Software is Always Vulnerable

- Program bugs
  - Software development errors including programming errors

- Software vulnerability
  - A weakness that may allow an attacker to develop and launch an attack

- Vulnerability can be introduced by different development stages
  - Requirements flaws
  - Design and architecture flaws
  - Infrastructure flaws
  - Implementation flaws
  - Integration flaws
  - Deployment flaws
Security Vulnerability and Attacks

- Zero Day vulnerability and attack
  Un-exploited and un-known security holes to vendors that can be developed into brand new attacks

- White-box vulnerability
  - A weakness that allows man-at-the-end attacks

- A security vulnerability is the intersection of three elements:
  - A system susceptibility or flaw
  - An attacker has access to the flaw
  - An attacker’s capability to exploit the flaw

- Attack Surface
  - The sum of the different points where an attacker can break a system
Challenges to design a secure system

- The system should be secure but ....
  - Be functional, usable and easy for users
  - Be within the computational, memory and power consumption budget of a device
  - Have a lifecycle – be manufactured, distributed, used and end of life
  - Be cost effective – cost significantly less than the asset to be protected
  - Fulfill time to market requirements

- Remain secure over the life cycle of the system
Economics of Security (2/3)

- Challenges to an attacker
  - Find a single point of failure of security
  - Cost of finding and reproducing attack should be much less than the reward
  - Depending on attack – reward ranges from sense of achievement to billions of dollars

- The attacker’s job is often much easier than the designer’s
  - The designer needs to make a complex system work all the time without any point of failure
  - The attacker just needs to find a single flaw as a start
Cost of finding problems per development stage

- Requirements: $1
- Design: $10
- Development: $100
- Testing: $1000

Architectural Review - Fast and Superior Results

- Requires half the time of the more commonly used penetration testing
- Typically takes only a few weeks to identify most security problems
- Provides far superior results and costs significantly less
Software Security Debt

- Requirement Debt
  - Miss security requirements
- Architecture Debt
  - Poor security architecture
- Design Debt
  - Poor security design decision
- Implementation Debt
  - Poor implementation including bad coding
- Test Debt
  - Lack enough security testing and security assurance
Hostile is a Normality, Trusted just is a State

Building a trustworthy application system is an evolutionary process to address attacks from one release to the next.
Connected and Intelligent Vehicle is the Biggest Target

- Hacking individual vehicles in local and remotely
- Hacking communications
- Hacking mobile devices
- Hacking the cloud servers
- Hacking and stealing data
- Hacking infrastructures
Cybersecurity Trajectory

00 Educating
A “new” industry’s cybersecurity cycle begins with security researchers raising industry awareness of vulnerabilities

01 Probing
Once academics expose vulnerabilities, elite hackers become attracted, searching for new technical challenges

02 Specializing
After elite hackers create working hacks, they develop highly specialized toolkits that enable unsophisticated users

03 Commercializing
In about three years, black-hat hackers determine how to monetize security flaws for eventual sale

04 Profiteering
Within 4-5 years, hacks are sold through openly illegal services to unlawful hackers and existing criminal organizations
1. Cloud planform and environment security threats
2. Network and communication security threats
3. Devices security threats
   - T-Box threats
   - IVI threats
   - Inside device network threats
   - OS threats
   - Sensor threats
   - Interface threats
   - PKI client threats
   - Data security threats
   - OTA security threats
4. External security threats
   - Mobile App security threats
   - Charging pile security threats
Intelligent Vehicle: Most Complicated Eco-System

Wire and wireless connectivity
- CAN bus / Ethernet / USB
- Bluetooth / Wifi / WAVE
- GSM / LTE / 4G / 5G
- Internet / ZigBee / DSRC

Increased communications
- V2V / V2X / V2P / V2M / V2C

Increased data gathering and managements
- Vehicle data / Driving data
- Driver data / User data
- Environment data / Application data
- Privacy data / Security data
- Insurance data / Warranty data

Increased functionalities
- Power train / body control
- Chassis control
- Infotainment / Living room capability
- Self driving L1-5
- OTA / mobile payment

Increased hardware & platforms
- ECU / TBOX / Many sensors
- Linux / QNX / Android / YunOS

Increased services and apps
- In vehicle / Mobile / Cloud
- OEMs / Tier +1 / 3rd Party
- Governments / Countries
- Insurance / Banks
Who Drives the Vehicle: Software

**Platform Software**
- Operating systems
- Framework / Middleware
- Cloud computing services
- Internet / Mobile networking

**System Software**
- Driving control systems
- Navigation systems
- Automatic driving systems
- Network communication systems
- Inspect and diagnosing systems
- AI and machine learning systems
- VR systems
- Data collect, storage and management systems
- Sensor software

**Application Software**
- Entertainment features
- Living room features
- Mobile office features
- Mobile payments
- Image processing

**Lines of Code**
- Space shuttle: 400,000
- Boeing 787: 14,000,000
- Today ICV: 100,000,000
- Tomorrow’s AV: 200,000,000

**Security Software**
- Secure environments
- Privacy enforcement
- Cryptographic software
- Data security software
- Security telemetry
Risks of Rubbish Security in Intelligent Cars

More and More Automotive Recall For Software Security Problems
Part 2: New Challenges and White-box Security
New Challenges to Information Security

Traditional Security Seriously Broken
As digital technologies become universal, they have transformed the people living and life, and business environment.
20 Years Ago: Trusted System is Main Stream

Traditional security is more about security of trusted environments
Connected Environments are Hackable

Cloud Computing Environments

Internet

UnTrusted

UnTrusted

Smart City

UnTrusted

UnTrusted

UnTrusted
Traditional Security: Seriously Broken (1/3)

Traditional security is not designed to counter today’s new and advanced threats. Why?

- **Sandboxing-based**
  - One of oldest security techniques and has been widely using to prevent traditional man-in-the-middle attacks not man-at-the end attacks.
  - It seeks to protect the host against hostile software, not the SW systems or applications and their data against the potentially hostile host.
  - It leaves us with a false sense of security: many threats cannot be addressed by the approach. For example, it does absolutely nothing to prevent massive surveillance.

- **Signature-based**
  - The long-standing blacklisting approach is losing the battle against new malware.
  - Right now, about more about 1 million new pieces of malware every day, and this will get much worse in the future.
  - Signature-based defenses are grossly insufficient.
Traditional Security: Seriously Broken (2/3)

- **Perimeter Oriented**
  - Perimeter oriented approaches concentrate on preventing or detecting threats entering networks of an organization, but perimeters are very porous these days.
  - Anything with an IP address can be a launch pad for attackers
  - Perimeter tools and security techniques were not designed to protect the data and against today’s advanced threats.
  - Within the perimeter, old security models are reactive. When you get past the perimeter, it’s no longer safe.
  - With the range of new use cases that need to be supported,
    - from BYOD to fixed function devices,
    - from accessing legacy web apps to new cloud-based app development and services,
    - IT is left with the challenge of working with a varied set of non-integrated tools while striving to achieve regulatory compliance and security at the same time.

- **Compliance oriented**
  - Compliance meets the requirements of auditors, or specific government mandates, rather than addressing the biggest current threats.
  - The danger is that we may mistake compliance with security standards for actual security.
  - They are two very different things, and some of them only deal with past threats.
Fixed Security

- A typical approach to security is to assume that the initial design will remain secure over time.
  - It treats security as a fixed target
  - Assuming that innovative attacks will not arise after deployment.

- Most security designs and implementations follow such a static deployment model, especially for hardware-based security.

- Once cracked, hardware security can’t be recovered quickly or cheaply.

- In reality, anything, including clever hardware, can be hacked given enough time and effort.

Therefore, new dynamic security approaches treat security as evolving and assume that security must be continually renewed, whether as part of ongoing policy or reactively.
New Fundamental Challenges to Information Security

Traditional security is more about security of trusted environments:
Stop any stranger to get into my house

White-box security and SW protection is more about security of un-trusted environments:
If a strange gets in my house, recognize and watch out, and protect valuable assets to prevent possible attacks in the house
Just Like Security and Protection in Museum
White-Box attacks are everywhere within un-trusted environments.
Man-In-The-Middle Attacks
(indirect & side-channel)

Black Box Attacks or Grey Box Attacks

Alice

Perimeter Defenses

Network

Bob

Trusted Inside of Black Box

- Alice and Bob each have exclusive control over their own computers
- No information leaves from or store into their computers without their approval
Major MITM Threats

Malware
- Adware
- Spyware
- Viruses
- Ransomware
- Trojan Horses
- Worms
- Rootkit
- Bootkits
- Bots
- Bugs
- Backdoors
- Logic Bomb
- Browser Hijacker

Illegal Access
- Eavesdropping
- Spoofing
- Buffer Overflows
- TCP/IP Hijacking
- Replay Attacks
- Phishing
- Sniffing

Other kinds
- Crypto Attacks
- Password Attacks
- DoS and DDoS
- Social Engineering
- Spamming
- Keyloggers
- Data Scraping
Attacking has much less limitation than protection

- Device and environment are un-trusted
- Attacker has direct access to the machine and software no matter whether it's running or not

**Attackers have open-end powers to do**
- Trace every program instruction
- View the contents of memory and cache
- Stop execution at any point and run an off-line process
- Alter code or memory at will
- Do all of this for as long as they want, whenever they want, in collusion with as many other attackers as they can find
White-Box Security Challenges

- **White-box**
  - Input / output
  - Timing analysis
  - Power analysis
  - Fault injection
  - Man-in-middle-attacks
  - SW vulnerabilities
  - Buffer overflow

- **Grey Box**
  - Timing analysis
  - Power analysis
  - Fault injection
  - Man-in-middle-attacks
  - SW vulnerabilities
  - Buffer overflow

- **Black-box**
  - Input / output
  - Timing analysis
  - Power analysis
  - Fault injection
  - Man-in-middle-attacks
  - SW vulnerabilities
  - Buffer overflow

- **Hardest**
  - Much more difficult to protect

- **Easiest**
  - Easy to protect

- **Strongest**
  - Debuggers
  - Emulators
  - Computation tracing
  - Decompiles
  - Profiling
  - IDA pro
  - Symbolic analysis
  - Malware
  - Other attack tools & methods
What Are the Threats?

Direct WhiteBox Attack

IDA Pro
HexRays
OllyDbg
LordPE
GDB
HIEW
HexEdit
VMware
QEMU

Colluding Attack

Differential Attack

version1
version2
time
Direct Analysis Attacks (Passive)

Static Analysis
- Disassembling
- Decompiling
- Hex Editing
- Binary Differing
- Program Graphing
- Similarity Comparing

Dynamic Analysis
- Debugging
- Using IDA Pro
- SMT/SAT Solving
- Combining Static & Dynamic
- Pattern Matching
- Colluding
- Program Slicing

- Emulating
- Tracing
- Profiling
- Memory Dump
- Monitoring
- Symbolic Execution
- Process Snooping
Direct Tampering Attacks (Active)

Static Tampering
- Data Lifting
- Code Lifting
- Binary Editing
- Function Replacing
- Program file Replacing
- Data file Replacing

Dynamic Tampering
- Using Debugger
- Using IDA Pro
- Combining Static & Dynamic
- Code/Data Modifying
- Function Call Shimming
- Branch Jamming
- Data Injection
- API Replacing
- Execution Alternating
- Dynamic Library Exploiting

Combine & Execute Static & Dynamic
Two Kinds of MATE Attacks

- Attacks by Human Hackers Directly
- Attacks by Intelligent Malwares and Botnets (Robot-Hackers) Directly

White-Box Environment (Hostile)
Software Protection Challenges

How to provide necessary trustworthy to software in the Un-Trusted Environment

Bob is the Attacker

Alice

Software

Network

Software
Both software security and software protection must become mainstream, not only in the commercial world, but also in the research community.

Software Security: More Than Vulnerability Check and Detection
```c
#include <stdio.h>

main() { /* Validate the users input to be in the range 1-10 */
    int number; int valid = 0;
    while( valid == 0 ) {
        printf("Enter a number between 1 and 10 -->");
        scanf("%d", &number);
        /* assume number is valid */
        valid = 1;
        if( number < 1 ) {
            printf("Number is below 1. Please re-enter\n");
            valid = 0;
        }
        if( number > 10 ) {
            printf("Number is above 10. Please re-enter\n"); valid = 0; }
    }
    printf("The number is %d\n", number); }
```

Important constants are exposed in memory

Function calls can be snipped and snooped

Operation can be modified statically and dynamically

Branches can be jammed dynamically

All vulnerabilities must be prevented by SW protection
- Session key is sent in the clear
- Content key is exposed on the client
- Content is exposed on the client
- Session and content keys can be extracted during use

All vulnerabilities can be prevented using White-Box Crypto
Specific Problem: Black Hole Effect

- The black hole effect occurs when part of the application is very secure but the rest is in the clear.
- Hackers mostly attack the boundary between the secure and the non-secure parts of a program.
Specific Problem: Black Hole Effect: How to Fix It

- To Fix the Black Hole effect
  - More lightweight obfuscation in the rest of the program
  - Faster generated code so that more security can be used in the white area
  - Transcoder Levels for low security on the rest of the application

- Blur the boundary between the secure and the rest of the program at low cost
White-Box security is new security paradigm well beyond traditional computer and network security.
Software Protection at All Levels

- Use software protection tools and libraries to make software self-protected at build-time
- Provide a comprehensive approach to software security
MULTI-LAYERED AND INTERLOCKED SW PROTECTION

- Protect application code against a collection of attacks
- Provides a multi-layered and interlocked defenses
- Flexible and modular to choose the right combination of defenses

Making security inseparable from your software
C/C++ Protection and Binary Protection

Transcoder
- Transcoder Front-End
  - Fabric ++
  - Cloaking Engine
  - Cloaked Fabric ++
  - Transcoder Back-End
- Protected C/C++ Source Code

C/C++ Source Code

Compiler & Linker
- WB Crypto Tool
- (Protected) Binary
- Binary Protection Tools
  - Secured Libs & Agents
  - Full Protected Binary

Native Execution Environment

Source Level Protection
- Source Level Protection

Binary Level Protection
- Binary Level Protection
Unified Cloaking Toolset

- **C/C++ Source Code**
  - Transcoder Front-End
    - Fabric++
  - Cloaking Engine
    - Fabric++
  - Transcoder Back-End
    - Cloaked C/C++ Source Code
- **Source Code in other languages**
  - LLVM Front-End Tools
    - LLVM IR
  - F2L / L2F Converter
    - LLVM IR
  - Binary Rewriting Tool
    - LLVM IR
  - LLVM Optimizer & Other Tools
    - LLVM IR
- **JavaScript/Binary Code**
  - LLVM Compiler & Linker
    - LLVM IR
  - Emscripten
    - Optimized & Cloaked asm.js, JS, Wasm
  - LLVM IR
  - C/C++ Compiler & Linker
  - Optimized & Cloaked Native Code
Diverse software instances are functionally equivalent but structurally and semantically diverse.

Each instance must be attacked separately by a skilled hacker.

Dramatically increases the work to create an automated attack tool.

The production of diverse instances is fully automated by the Cloakware tool chain.

Application software

Random seed values

Diversified instances of application software
Software Diversity

- All program Constructs can Be Diversified
  - Randomly Chosen:
    - Order & program Layout
    - Function Families
    - Constants
- Seeded Build
  - Reproducibility
- Diversity Control and Opportunities
  - On the source level
  - At the compilation time
  - On the library level
  - At the link time
  - On the binary level
  - Combination above
- Static and Dynamic Diversity

Many software protection techniques can make own contributions to software diversity
Cloakware Securing Software - Built on Core Technology

Cloakware for Software Protection

Application Protections

API Protection
Node Locking
Jailbreak & Root Detection
Diversity & Renewability
Anti-Hooking
Java Access Control
Anti-Tamper
Anti-Reverse Engineering

Code Obfuscation
Data Obfuscation
Function Obfuscation
Control Flow Obfuscation
Secure Inlining & Merging
Control Flow Integrity
Code Entanglement

Binary Protection
Application Signing
Platform Flexible Integrity Verification
Anti-Debug
Secure Loader
Platform Flexible Fingerprinting

aphy
White Box Cryptography
- RSA
- AES
- ECC
- 3DES
- Whitebox PRNG
- Secure Store

Security Tools
- Secured Libraries (FlexLib)
- Smart Assembly
- Key Transformations
- Secure Code Injection
- Security Plugins
- Secure Heap
## Defense in Depth

<table>
<thead>
<tr>
<th>Technology</th>
<th>Prevent Analysis</th>
<th>Prevent Tampering</th>
<th>Foil Automated Attacks</th>
<th>Renew and Diversify</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Static</td>
<td>Dynamic</td>
<td>Static</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Data Flow Transforms</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Control Flow Transforms</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>White-box Crypto</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Secure Store</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Integrity Verification</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Anti-Debug</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Code Encryption</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
Deployments Are Rarely Simple

- **Multiple Languages**
  - C, C++, Java, C#, .NET, JavaScript, Flash, Ruby, Perl, Ajax

- **Heterogeneous System Run Environments**
  - Android: Linux, Native & Dalvik VM
  - BluRay Disc: BD+ VM & Native & BD-J
  - WinMobile: C#, Native

- **Multiple Platforms**
  - Adobe Flash Access: PC, Mac, QNX, Android
  - Apple iTunes: Mac, Win, iOS
  - Comcast Xfinity: iOS, Android
  - CA: ST40, MIPS, x86, Amino, Broadcom

Security must balance with constraints, in particular, performance
Security is very important, but hard to measure.

Security tends to get focus after a successful attack in the field.

Until then, size and speed concerns dominate.

Adding security to part of an application:

10x bigger and slower  →  Usually acceptable

100x bigger and slower  →  Sometimes acceptable

1000x bigger and slower  →  Almost always unacceptable

All security measures must give enough improvement to justify their costs.
A much bigger security challenge
AI-Enabled Everything

Significant Success

- Big Data Processing and Data Mining
- Speech Recognition
- Face Recognition
- Fraud Detection
- Financial Trading
- Machine Translation
- Image Processing
- Spam Filters
- Search Engines
- Predictive Analysis of Consumers
- Intrusion Detection

Profoundly Effected

- Multimedia
- Healthcare
- E-commerce
- Knowledge Discovery
- High Performance Computing
- AI-enabled IT
- AI-enabled Robotics
- AI-enabled Platforms
- Al-enabled Cloud Services
- AI-enabled Devices
Intelligent Vehicles Depend on AI

Functionality, Safety, Reliability, Security, Usability

Driving and Speed Control
Environment Assessment & Processing
Collect, Transfer and Process Data
Telematics and Diagnostic
Navigation & Infotainment Control
Security Problems of AI and ML

- Adversarial Machine Learning
  A malicious adversary can carefully manipulate the input data exploiting specific vulnerabilities of learning algorithms to compromise the whole system security.

- Malicious Use of AI and ML
  Attackers leverage the ability of AI and ML and speed up the development and launching of attacks for which current technical systems and IT professionals are ill-prepared, and impacts and scopes can be much serious.

- Directly Attacking AI and ML
  Within a MATE environment, hackers can access directly, and analyze the AI and ML software to understand algorithms, models, classifiers, solvers, and inner workings associated with its designs, functionalities, features, and capabilities; and modify the code, inputs, outputs and other critical assets of the AI and ML system to alter the application to achieve their own hacking purpose and to profit by breaking the original system intent.
Adversarial Machine Learning (1/2)
Adversarial Machine Learning (2/2)
Malicious Use of AI and ML

One recommendation from the report is that some AI and ML research may need to be kept secret due to the dual-use nature of AI and ML technology. How?

- **Expansion of existing threats**
  The costs of attacks may be lowered by the scalable use of AI systems. The set of actors can be expanded to carry out particular attacks and the set of potential targets.

- **Introduction of new threats**
  New attacks may arise through the use of AI systems to complete tasks that would be otherwise impractical for humans. In addition, malicious actors may exploit the vulnerabilities of AI systems deployed by defenders.

- **Change to the typical character of threats**
  There is reason to expect attacks enabled by the growing use of AI to be especially effective, finely targeted, difficult to attribute, and likely to exploit vulnerabilities in AI systems.

New report on “The Malicious Use of Artificial Intelligence: Forecasting, Prevention, and Mitigation” was released by late March by 26 experts from 14 different institutions, warning changes in the landscape of security threats:
Directly Attacking AI and ML
Connected Application Central based Security Model

Trusted model to address both man-in-the-middle and man-at-the-end attacks
The security lifecycle of a digital asset application mandates protection from creation, through distribution and then ultimately consumption from being deployed in the field.
Typical approach to security is to assume that the initial design will remain secure over time

- Anything will be hacked given enough time and effort
  - Set top boxes, PC apps, Mobile devices, CE devices, TCM, TEE

- Application or content owners want to know, “What is your security strategy”? 
  - How will you limit potential damages if there is a breach?
  - What is your renewability strategy?
The Result of Static Security

Static Security Compromised

Installed Base Security

Time

Breach
Dynamic Security is a security model that enables the protection of digital assets against unauthorized use through the upgrade and renewal of the underlying security in the field.

- **Proactive prevention**
  - Monitor hacker channels to understand attack techniques and methodologies
  - Apply security updates to reset the hacker’s clock

- **Reactive reduction**
  - Limits the impact of a breach before it has a significant impact

- **Benefits:**
  - Disrupt potential hacks before they happen
  - Mitigate impact of a security breach
  - Minimal disruption of business
Static Security vs Dynamic Security

Once static security breaks, the entire security is gone and hard to be restored.

Once dynamic security breaks, the security can be renewed and restored immediately in a planned way.
Minimize scope of attack -- Prevent automated attacks

Provide rapid recovery in the event of an attack

Make the business unattractive to the hacker
Dynamic Security and Lifecycle Management

Pre-Launch

Product Development
Optimize Security Design

Security Design

Initial Attack Resistance

Post-Launch

Watch And Defend
Attack Monitoring

Mitigation Planning
Attack Analysis

Renewed Attack Resistance
Countermeasures Design & Dev

Active Cloak Server

Active Cloak

Active Cloak Server
A great opportunity for researchers to discover and develop measurement model and metrics on SW security and protection.
New View of Information Security

Static Security

Security to Man-In-The-Middle Attack

Security to Web Browser

Dynamic Security

Security to Man-At-The-End Attacks

Security to Remote Attack
Security vs. Practice

Technology Gap

- Homomorphic Transformation
- White-Box Crypto
- Other SW Protection

Full Homomorphic Encryption
Indistinguishability Obfuscation

Cryptography and SW protection research can make good contributions

Highly Practical

Much Less Practical
Attack Effectiveness Against SW Protection

- High
- Low
- High

Analysis Precision

Defenses

Attacks

Attack Effectiveness
Threat & Attack Index

Successful Attack

- Understand code
- Identify and break defenses
- Locate and find assets
- Leverage vulnerabilities
- Modify code and data
- Develop an attack
- Launch the attack

No Defenses or Defenses are broken
Effective Defenses

- Make attacks more difficult, less effective
  - Make understanding harder
  - Prevent asset identification
  - Prevent code and/or data tampering
- Monitor, detect, and mitigate attacks
- Renew security: ongoing security lifecycle management
It is very difficult by adapting any existing theories and methods to develop commonly acceptable metrics on the effectiveness of SW protection.

- Existing software complexity techniques and methods have very little value for resolving this problem.
- Current computation complexity theory cannot apply easily and directly to develop a formal model for such a measurement.
- Cryptographic analysis methods on black-box security are not applicable well for many cases.
Some Interesting Observations (Can be a long list)

- SW protection needs to prevent all attacks if possible but attacking only needs to find one place to break.
- There is no single protection can stop all attacks. Instead, we have to layer and combine different protection techniques into a protected and interlocked security maze.
- More [less] complicated/protected software doesn’t mean more [less] secure
- Static measurement is not enough to address security dynamics
- Attacking mainly is a manual process. How to measure and factor the effectiveness of attacks by different skilled attackers?
- Security has to deal with unknown attacks in the future? How to factor uncertainty nature of any new attacks?
- Perfect security does not exist! SW security must be relative in a real environment with limits!
- Strong security needs renewability! How to measure renewability?
Automotive industry is used to measure and control everything that have to be put into a car.

Of course, they would like to measure SW protection and have a qualification control how security for used SW protection techniques.
Software security immunity is a concept that needs to be developed as a framework lay-outing foundation and relationship between different SW security issues.
In biology, **immunity** is the balanced state of multicellular organisms having adequate biological **defenses** to fight **infection**, **disease**, or other unwanted biological **invasion**, while having adequate tolerance to avoid **allergy**, and **autoimmune diseases**.
The immune system is a host defense system comprising many biological structures and processes within an organism that protects against diseases.
Layered self-defense
- First line of defense: Innate immune subsystem
- Second line of defense: adaptive immune subsystem
  - More immune subsystems: nature & artificial, passive & active

Self-evolving
- Every time that antigen invades the body, the body remembers (memory), and an appropriate and specific response is produced by the host immune cells and antibodies so that it can launch a faster, stronger and longer-lasting counterattack if such an antigen reappears.

Specificity
- Response to one organism does not give protection against another unrelated organism

Diversity and Versatility
- Millions of antigens in the environment can pose a threat to health. Over a normally lifetime, an individual encounters only a fraction of that number. It must be ready to confront any antigens at any time. With diversity and versatility of immune system, each individual has own unique immutable capability.

Tolerance
- The immune response targets foreign cells and compounds, but it generally ignores normal tissues in the body.
Any connected system targeting consumers’ market is a hostile environment where software is vulnerable for various types of attacks.

Software (platform layer, system layer and application layer) must build up **security immunity** to protect the software against external attacks.

Software security immunity needs to define a set of **properties** and **attributes**.

Software security immunity needs a **framework** to structure, orchestrate, and coordinate security **elements** to prevent and fight attacks effectively.

Software security immunity needs a **strategy** to manage and maintain security during its lifetime.

Software Security immunity requires right **technologies**, **methods**, **solutions** and **tools** to provide security capabilities while the secured software to be designed, developed, implemented, deployed, and maintained and renewed.
SW Security Immunity (SWSI)

- **Perimeter Defense**
  - Access control
  - Malware detection
  - Antivirus
  - Firewall
  - Authentication
  - Authorization
  - Intrusion detection
  - Cryptography
  - Vulnerability detection
  - Memory safety

- **Privacy**

- **Integrity**
  - Data integrity
  - Code integrity
  - Execution integrity
  - AI integrity
  - System integrity
  - Environment Integrity

- **Self-Defense**
  - Data and code transformation
  - Program obfuscation
  - Secure storage
  - White-box crypto
  - Code encryption
  - Trusted AI and ML
  - API protection
  - Anti-debug
  - Anti-hooking
  - Node-locking
  - Program entanglement & interlocking

- **Diversity & Renewability**
  - Hacking monitoring
  - Security analytics
  - Telemetry
  - Ongoing verification

- **Attack Detection**
  - Secure OTA
  - Dynamic Policy Update
  - On device response
  - Policy based response
  - Renewable security

- **Response & Mitigation**

- **“Innate” SWSI**

- **Adaptive SWSI**
Part 3: **White-box Security Patterns**
Software security now is an art not a science. Pattern abstraction is one of valuable steps forward to scientific foundation.
WhiteBox Security Patterns

- Abstract and define white-box computing problems (vulnerabilities, threats and attacks) and establish the security solutions that defend against them
  - Develop a small and finite set of WB computing security patterns
  - Easy to understand and adapt in real world

- Create a new common language for software security and protection to
  - Provide an effective tool to promote software protection technology
  - Provide a foundation for software protection evaluation model and methods.
  - Make it much easier to engage the wider academic community, generate more research attentions and create generic mindshare

- As a reference used by security professionals and ultimately would become the secure application guidelines
  - The security patterns should be used to create a set of security architecture and design guidelines to the security professionals and security system designers
White-Box Security Patterns: Present & Future

- **Well Understood WBC Patterns**: (more about single environment)
- **New WBC Patterns**: (for connected multiple environments)
- **Advanced WBC Patterns**: (Covered by next generation of technology and products)
Direct Attack Points (Just Examples)

Producing Software *(Trusted Environment)*

- Source code
- Compiling & Linking
- Executable

Distributing Software *(via 3rd Party Environment)*

- On-line Download
- Build-in
- Pre-Install
- Executable

Running Software *(White-Box Computing Environment)*

- Data and data flow
- Operations
- Program decision and property
- Functions and invoking

- Executable
- Other Applications

- Tampering Attacks
- Automatic Attack
- Attack Tools
- Cloning Attack
Well Understood WB Security Patterns

- **Primitive Patterns**
  - Pattern 1: Homomorphic data transformation
  - Pattern 2: Protecting program decision and property
  - Pattern 3: Function boundary concealment
  - Pattern 4: Control flow obfuscation
  - Pattern 5: Execution flow integrity
  - Pattern 6: White-Box cryptography
  - Pattern 7: Program integrity verification
  - Pattern 8: Anti-debug
  - Pattern 9: Secure loader
  - Pattern 10: Dynamic code decryption
  - Pattern 11: SW anchoring

- **Abstract Patterns**
  - Pattern 12: Software diversity

- **Derived Patterns**
  - Pattern 13: Reinterpretation
  - Pattern 14: Shim detection
Description in Details for Three White-Box Security Patterns
**WB Security Patterns**

- **Primitive Patterns**
  - **Pattern 1:** Homomorphic Data Transformation
  - Pattern 2: Protecting program decision and property
  - Pattern 3: Function boundary concealment
  - Pattern 4: Control flow obfuscation
  - Pattern 5: Execution flow integrity
  - Pattern 6: White-Box cryptography
  - Pattern 7: Program integrity verification
  - Pattern 8: Anti-debug
  - Pattern 9: Secure loader
  - Pattern 10: Dynamic code decryption
  - Pattern 11: SW anchoring

- **Abstract Patterns**
  - Pattern 12: Software Diversity

- **Derived Patterns**
  - Pattern 13: Reinterpretation
  - Pattern 14: Shim detection
Homomorphic Data Transformation: Security Context
Homomorphic Data Transformation: Security Problem (1/2)

- At runtime, data frequently exists in a program or file in various classes of storage for white-box exposure:
  - In registers
  - on the stack, the heap or disk
  - other forms of secondary storage etc

- Computable data stored in different storage forms or transferred from different sources may have different vulnerabilities, but a contributing factor common to them is the well-known layout of data while they are processed by a program
- Data can be transferred dynamically via network connections to local device so that they can be accessed by local program.

- Once the attacker reaches a data asset, that asset succumbs completely because the data is stored using conventional formats.

- The attacker will know how to discern the object’s value and assign to it a properly hacked value.

- These kinds of storage technology normally have little protection against white-box attacks.
Modern computer systems provide an open & common computation space.
Computational data is a crucial asset needing protection.
Both the original values of computational data, and the computations on it, must be hidden to protect against reverse-engineering and subsequent code compromise.

- Using static tools such as program analyzers, binary editors and disassemblers.
- Using dynamic tools such as debuggers, logic analyzers and emulators.

Transforming of data, computations and data flow is an essential first step in HO.
Homomorphic Data Transformation: Principle

Value Coding

encoding of X
\( X' = C_X(X) \)

encoding of Y
\( Y' = C_Y(Y) \)

encoding of OP and result value Z
\( Z' = C_Z(F(X,Y)) = C_Z(F(C_X^{-1}(X'), C_Y^{-1}(Y'))) \)

The implementation computes in the Transformed Space without encoding/decoding operations.

The green blob is a homomorphism with 3 connecting encodings (for 2 inputs and 1 output). Multiple distinct homomorphic blobs connect at the I/O points, making the obfuscated code a homomorphic network.

New code for transformed computation (OP) and its result value (Z) in transformed form. Original X, Y and OP are disappeared.

‘Smooth’ Common Space: \( Z = F(X, Y) \)

Original data flow

Cloaked data flow

Relationship between smooth and transformed value

Transformed Space: \( Z' = F'(X', Y') \)
Homomorphic Data Transformation: In General

- Mathematical transformations on:
  - Data Values
  - Data Locations
  - Data Operations
- Many Transform Families
- Randomness
- Random seeds support repeatability
- Must balance security vs. performance to fit the application

**Original Data Flow Graph**

**Data Transformations**

**Transformed Data Flow Graph**

**Mapping**

Original data, value, operation and data flow are hidden after data transformation.
Homomorphically Transformed Computation Space

Homomorphic transformation of data and computation space is fundamental to homomorphic obfuscation.
Many to many mappings between original and transformed data and code make reverse engineering difficult

(*NP-complete fragment recognition problem*)

<table>
<thead>
<tr>
<th>Original code segment</th>
<th>Sample 1</th>
<th>Sample 2</th>
<th>Sample 3</th>
<th>Sample 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Transcoder transforms</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x' = 5x + 7$</td>
<td>$x' = 5x + 7$</td>
<td>$x' = 5x + 7$</td>
<td>$x' = 5x + 7$</td>
<td>$x' = 5x + 7$</td>
</tr>
<tr>
<td>$y' = 5y + 10$</td>
<td>$y' = -10y + 10$</td>
<td>$y' = -5y + 11$</td>
<td>$y' = -5y + 11$</td>
<td>$y' = 5y + 10$</td>
</tr>
<tr>
<td>$z' = 5z + 17$</td>
<td>$z' = 10z + 4$</td>
<td>$z' = 5z + 3$</td>
<td>$z' = 5z + 3$</td>
<td>$z' = 5z - 18$</td>
</tr>
<tr>
<td><strong>Transformed code segment</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$z' = x' + y'$</td>
<td>$z' = 2x' - y'$</td>
<td>$z' = 2x' - y'$</td>
<td>$z' = 2x' - y'$</td>
<td>$z' = x'$</td>
</tr>
</tbody>
</table>
Homomorphic Obfuscation vs. Typical Obfuscation

Normal (Naïve) Program Obfuscation

- Original Program
- Obfuscator
- Obfuscated Program

Original and Obfuscated programs are in the same original computation space.

HH Program Protection

- Original Program
- Homomorphic Hardening Tools
- HO-Protected Program
- Original version
- HO Program version 1
- Protected computation space 1
- ... Protected computation space n
**Primitive Patterns**

- Pattern 1: Homomorphic data transformation
- Pattern 2: Protecting program decision and property
- Pattern 3: Function boundary concealment
- Pattern 4: Control flow obfuscation

**Pattern 5:** Execution flow integrity

- Pattern 6: White-Box cryptography
- Pattern 7: Program integrity verification
- Pattern 8: Anti-debug
- Pattern 9: Secure loader
- Pattern 10: Dynamic code decryption
- Pattern 11: SW anchoring

**Abstract Patterns**

- Pattern 12: Software Diversity

**Derived Patterns**

- Pattern 13: Reinterpretation
- Pattern 14: Shim detection
Basic blocks are primary components of program execution flow. Flow dependency between those basic blocks is statically fixed.

Control flow obfuscation provides only for hiding the original intent of the control flow, but cannot guarantee execution flow integrity.

The protection of execution flow of a function requires to resolve the following two problems:

- Transform control flow and make the control flow hard to be analyzed and extracted statically and dynamically.
- Transform execution flow of a function so that the flow cannot be tampered easily and can be detected and mitigated if it is tampered.
Execution Flow Integrity: Security Intent

- Extend original execution flow with history dependency based on original execution order of a function
  - For each particular flow from one basic block to another block, inject a pair of encode and decode and necessary temporary variables to interlock the flow
  - The original control-flow is transformed into a data directed control-flow by injected history dependency
  - The extended execution order is no longer static and must be determined at run-time by the computation of history dependency relationship
- Data transformation can be used to protect the history dependency computation
- Any tampering attack to history dependency will result wrong execution flow
Execution Flow Integrity: Solution – Control Flow Flattening

Original Program Flow

Control Flow Flattened Program
Execution Flow Integrity: Solution - History-Dependent Transforms

Switch (case)

1. case = b
2. case = D(d)
3. case = D(c)
4. case = D(a)

b = D(b) → b = E(b) → d = E(d)

a = 99 → a = E(a) → a = 99

c = E(c)

1 = 3 = 7 = 2

HD(a) → HD(b) → HD(c) → HD(d)
WB Computing Security Pattern

- **Primitive Patterns**
  - Pattern 1: Homomorphic data transformation
  - Pattern 2: Protecting program decision and property
  - Pattern 3: Function boundary concealment
  - Pattern 4: Control flow obfuscation
  - Pattern 5: Execution flow integrity
  - **Pattern 6:** White-Box cryptography
  - Pattern 7: Program integrity verification
  - Pattern 8: Anti-debug
  - Pattern 9: Secure loader
  - Pattern 10: Dynamic code decryption
  - Pattern 11: SW anchoring

- **Abstract Patterns**
  - Pattern 12: Software Diversity

- **Derived Patterns**
  - Pattern 13: Reinterpretation
  - Pattern 14: Shim detection
Modern cryptography is one of the most fundamental technologies adopted for traditional security problems.

- Art
- Solid Security Primitives and Protocols
- Cryptography Past 90 years
- Science
White-Box Cryptography: Security Problem – Key and Valuable Assets

- Software keys can be
  - generated using high-quality pseudo random number generators (PRNG)
  - securely stored

- Sooner or later the key is *used* and the following events occur:

  - Extract key from storage (encrypt, deobfuscate)
  - Data
  - Decryption
  - Key
  - Key is easily extracted by white-box attacker
We are now forced to defend against white-box attackers who are strictly more powerful than classic black-box and grey-box attackers.

How can a secret key be used in a cryptographic algorithm without being exposed in the context in which it is attacked in white-box fashion?

Existing cryptographic security proofs from the black-box and grey-box attack context simply don’t carry over to the white-box context.

*It is broken!*
The fundamental security intent of white-box cryptography is to make the recovery of the key in the white-box context at least as difficult, mathematically, as in the black-box context.

Stated in another way, this pattern is to transform a key such that attacking within the white-box context offers no advantage to attacking in the black-box context.

Black-box cryptographic security can be truly guaranteed within white-box context and even improved further if possible.

Ideally, we should develop White-Box friendly ciphers with strong security for both black-box and white-box contexts.
White-box cryptographic methods use homomorphic transformations.

White-box cryptography ensures that input data, keys, intermediate results and output data are protected at all times by using homomorphic transformations.
GENERAL REQUIREMENTS

- **Key Mode**
  - Fixed Key
  - Dynamic Key

- **Re-key/upgradable in the field**

- **Security Assessment**
  - All known WB attacks assessment
  - Diversity assessment
  - Possible unknown attacks assessment

- **Implementation Assessment**
  - Complexity
  - Speed
  - Memory size
White-box AES
- 128 and 256-bit keys
- ECB, CBC, OFB and CTR modes
- A variety of size/speed/security trade-offs

White-box RSA
- Fixed keys up to 16384 bits
- Dynamic keys up to 4096 bits
- OAEP and PKCS #1 v1.5 encryption padding
- PSS and PKCS #1 v1.5 signature padding
- Integration with MD5, SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512

White-box ECC
- Support for prime field curves with optimized support for NIST curves
- Fixed and dynamic-key sign/verify (ECDSA), dynamic encryption/decryption (EC El-Gamal)
- Integration with MD5, SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512

DES/TDES
- A variety of size/speed/security trade-offs

White-box SHA2
- Support for SHA-224 and SHA-256
- Protection of message, digest and intermediate calculations
Irdeto WB crypto technology can countermeasure and mitigate all known WB attacks:

- Plain input attack
- Plain output attack
- Replay attack
- Code/data lifting attack
- Key-update attack
- Collision attack
- BGE attack
- DFA attack
- DCA attack
- Affine attack
Thank You!

Questions?

www.irdeto.com | yuan.gu@irdeto.com