The Industrial Challenges in Software and Information Protection

Yuan Xiang Gu
Co-Founder of Cloakware
Chief Architect, Irdeto
Guest Professor, Northwest University

The 8th International Summer School on Information Security and Protection
July 17-21, 2017
Agenda

- Myself Briefing
- Irdeto Overview
  - Who are we, what do we do, and where we are evolving
- Part 1: Trends in Security Threats
- 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
  - Web Application Security Challenges
  - Connected Application central based Security Model
  - Software Security Lifecycle and Digital Asset Protection
  - New View of Information Security and New Research Opportunity
- Part 3: White-box Security Patterns
  - Introduction to WB Computing Security Patterns
  - WB Computing Security Pattern Description in Details
- Summary
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
2007 - present: Chief Architect, Irdeto Canada,
  - leading security research and collaboration with universities worldwide
2011 - present: Guest professor of Northwest University, China
Where is Irdeto Canada

It is really

COOL,

COOL,

COOL!!!
Intro to Irdeto

Who are we?
What do we do?
Where we are evolving?

cloakware®
Irdeto is part of the Naspers Group (NPN.SJ)

Naspers is one of the world’s largest technology investors

Revenue
US$ 12.2bn
+22% YoY
(+6%)

Profit
US$ 1.2bn
+49% YoY
(+21%)

Irdeto is the cybersecurity unit of the Naspers group

Cloakware is the software security brand for Irdeto’s IoT offerings
For nearly 50 years, Irdeto has worked with software application providers, connected device manufacturers, pay-media operators and content creators to secure their products and businesses.

Now turning to IoT, Irdeto believes that privacy and protection against cyber criminals is fundamental to building a healthy and safe digital future.

#1 in software security in pay media

+5 billion devices & applications secured

+191 million cryptographic keys generated and under management

Serving 350 clients worldwide

571 patents & 522 patents pending

+1000 security expert employees

20 locations covering 6 continents
Part 1: **Hacker Trends**

Days of hacking games and movies are over...

... Attacking business is the new trend!
"As the chairman pointed out, there are now computers in everything. But I want to suggest another way of thinking about it in that everything is now a computer: This is not a phone. It’s a computer that makes phone calls. A refrigerator is a computer that keeps things cold. ATM machine is a computer with money inside. Your car is not a mechanical device with a computer. It’s a computer with four wheels and an engine… And this is the Internet of Things, and this is what caused the DDoS attack we’re talking about."

– Bruce Schneier
Speaking before Members of US Congress
Nov 2017

“The Internet era of fun and games is over”
Smart Everything: Can Secure Everything?
IoT – How to Fail at Security

Stranger hacks family's baby monitor and talks to child at night

By CHANTE OWENS  November 3, 2016

IoT security camera info in seconds of plugging it in

It took a mere minute and a half for an intruder to find out it was "in use" with malware.

Z-Wave Alliance Announces New Security Requirements for All Z-Wave Certified IoT Devices

The Alliance Board of Directors has voted to mandate all devices receiving Z-Wave Certification after April 2nd, 2017 to include new advanced Security 2 (S2) framework

Hackers found 47 new vulnerabilities in DEF CON devices

The results from this year’s IoT hacking contest are in and it’s not a pretty picture
Philips Hue – Malware

Philips Hue – Home blacked out by malware researcher

By Sal Cangeloso on August 18, 2017

107 shares

Philips Hue bulbs, home blacked out by malware researcher

By Sal Cangeloso on August 18, 2017

107 shares
Automotive – Becoming a Favorite Target

- Local attacks
- Remote attacks
- Personal Data Theft
- Software bugs
- Architectural defects
2010
Wifi laptop remote control

2010
OnStar remote control

2015
Dealer malware propagation

2015
Key fob replicator

2015
Jeep hack

2016
Nissan Leaf mobile app

2014
Remote control through Zubie dongle

2015
OnStar unlock / start takeover

2015
Tesla WebKit hack

2015
Corvette insurance dongle

2015
BMW unlock

2016
Tesla Wifi and Android App

Automotive Cybersecurity History
Cybercrime has evolved from single hackers into resilient highly skilled organizations performing global cyber attacks

- 38.5% of firms have experienced a cyber attack in the past 12 months
- 21% of these attacks had a cost higher than 5 million EUR

(Source: Marsh report September 2016)
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

- **2016 Klinikum Arnsberg**
  - Germany

- **2016 Methodist Hospital**
  - USA

- **2016 Healthcare providers pay USD $6B annually to ransomware**

- **2016 Microsoft Presbyterian**
  - USA

### Healthcare Providers Affected

- **2016 Hollywood Presbyterian**
  - USA

- **2016 Lukas Hospital**
  - Germany

- **2016 Chino Valley Medical**
  - USA

- **2016 TeslaCrypt | 777 Xorist | Cerber**
  - GhostCrypt | SamSam
  - CryptoLocker
  - MSIL/Samas | Locky

- **2016 Kansas Heart Hospital**
  - USA

- **2016 DeKalb Health**
  - USA

- **2016 Ottawa Hospital**
  - Canada

### Ransomware Analysis

- **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
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?
Increased Connectivity means Increased Risk – and Different Attack Type

**TRENDS**
- Open systems, open source
- More third party applications, developer tools
- Regulatory compliance and third party licenses
- High value assets are now “connected”
- More applications, more access and private user data

**RISKS**
- More attack vectors
- Increased attacker incentives
- Greater Insider threat
- Device revocation
- Automated attacks

**EXPOSURE**
- Slower market adoption
- Financial loss
- Brand erosion
- Lost shareholder value

© 2017 Irdeto. All Rights Reserved. – www.irdeto.com
Problem: All software is vulnerable

- Advances in debugging and reverse engineering techniques have empowered increasingly more capable and tech-savvy hackers
- Unsecured software is as readable as a book – IP and critical algorithms are simple for hackers to access and exploit
- Open source software and hacker collaboration compound the problem, providing “easy learning”
- Hacking is a business - Hackers make their profit by scaling and selling modified versions of applications
Challenges to design a secure system

- The system should be secure but ....
  - Be 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
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
Vulnerability Does Always Exist

- **Vulnerability**
  A weakness which allows an attacker to develop and launch an attack

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

- **Attack Surface**
  The sum of the different points where an attacker can break a system

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

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

- 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
Planning an Attack on an IoT Target

- **Attack Points:**
  - Device (Receives the most focus)
  - Smartphone app
  - Communications and connection points
  - Other things the device connects to, like your router, your network, etc
  - Cloud (via the Internet)

- **Phases of an attack**
  - Investigation
  - Leverage a weakness
  - Peel the onion
  - Rinse and repeat
  - Launch an attack
Part 2: New Challenges and White-box Security
New Challenges to Information Security
Technologic Impacts to Modern Business Model

Information Security

User Model

Mobile Connectivity (including Consumer and Enterprise Devices)

On-line Business (anywhere, anything, anytime, any-device)

Computing Paradigm (including Cloud Computing and HTML5)

Internet (including Internet of Things and Social Network)
As digital technologies become universal, they have transformed the people living and life, and business environment.
Un-trusted environments are everywhere and even becoming more dominated in digital world.

Persistent security on un-trusted environments is becoming #1 concern.

Cloud Computing Environments

Consumer Devices & Home Networks & Internet of Things
Advanced Persistent Threats (APT or APA)

You are compromised! But you just don’t know it!!

1. Collect intelligence
2. Find a point of entry
3. Call home
4. Search for data/assets
5. Move throughout the network
6. Extract data

Targeted
Persistent
Goal-Oriented
Connect to home

Patient
Insider Threats - No longer an Incident, It’s a Big Problem

- Two categories of insider tcareless userhreats (ITS)
  - By [ex-]employees or associates of an organization who either maliciously or accidentally take action that put their organizations and data at risk.
  - By outsiders who have obtained the legitimate credentials needed to gain access and conduct malicious activities that cause operational harm and steal data using APT.

- Insider threats landscape (2017 report)
  - Top insider threats
    - Inadvertent data breaches (careless user): 71%
    - Negligent data breaches (user willfully ignoring policy): 68%
    - Malicious data breaches (user willfully causing harm): 61%
  - Data most vulnerable to insider attacks
    - Customer data: 63%
    - Financial data: 55%
    - Intellectual property: 54%

- Black-box context is facing new security problems that traditional security models and technologies cannot offer sufficient solution.
- There is no system can be fully trusted and secured.
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.
While information security risks have evolved and intensified, security technologies and strategies have not kept pace.

Today, organizations often rely on yesterday’s security strategies to fight a largely ineffectual battle against highly skilled adversaries who leverage the threats and technologies for tomorrow.

The sophisticated intruders are bypassing outdated defenses to perpetrate dynamic attacks that are highly targeted and difficult to detect.
Once a targeted attack is accomplished and the network is breached, there is nothing to stop the damage.

Organizations are still focused on stopping the landing point and not on what they must do.

What’s needed

- A new model of information security that is driven by knowledge of threats, assets, and the motives and targets of potential adversaries.

- A new understanding that an attack is all but inevitable, and safeguarding all data at an equally high level is no longer practical.

- Pioneering technologies, processes and a skill set based on counterintelligence techniques.
New Fundamental Challenges to Information Security

Traditional security is more about security of trusted environments

White-box security and SW protection is more about security of un-trusted environments
WhiteBox Attacks in Real World

White-Box attacks are everywhere within untrusted environments
Cryptographic Assumption and Traditional Attacks

Black Box Attacks or Grey Box Attacks

Man-In-The-Middle Attack
(Indirect, side-channel)

Perimeter Defenses

Network

Trusted Inside 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

Alice
Software

Bob
Software

Alice and Bob each have exclusive control over their own computers.
No information leaves from or store into their computers without their approval.
White-Box Attacks

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

Attacking has much less limitation than protection
- Device and environment are untrusted
- Attacker has direct access to the machine and software no matter whether it's running or not

Bob is the Attacker

Man-At-The-End Attack

Network

Alice
Software
Just Like Security and Protection in Museum
White-Box Security Challenges

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

- **Grey Box**
  - Debuggers
  - Emulators
  - Computation tracing
  - Decompiles
  - Profiling
  - IDA pro
  - Symbolic analysis
  - Malware
  - Other attack tools & methods

- **Black-box**
  - Easy to protect
  - Weakest
  - Attack Effectiveness
  - Strongest
  - Much more difficult to protect
What Are the Threats?

- Direct WhiteBox Attack
- Colluding Attack
- Differential Attack

Tools:
- IDA Pro
- HexRays
- OllyDbg
- LordPE
- GDB
- HIEW
- HexEdit
- VMware
- QEMU

version1
version2
time
Two Categories of White-Box Environments

- Attacks by Human Hackers Directly
- Attacks by Malwares and Botnets (Robot-Hackers) Directly
Perimeter Defenses Do Not Prevent White-Box Attacks Today

Traditional and classic computer and network security technology only provide perimeter defenses

- Firewall
- Authentication (VPN, SSL, …)
- Intrusion detection
- Malware detection and anti-virus
- Cryptography (Black box)

- Physical security
- Secure operation systems
- Software vulnerability check
- Identity management
- Trusted computing
PROBLEM
Perimeter security is invariably bypassed once hackers have physical access
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.
White-Box Vulnerabilities – Example 1

```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 -->\n");
        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
All vulnerabilities can be prevented using White-Box Crypto
Who Are the Hackers?

- Unsophisticated Attackers
- Organized Crime
- Nation States
- Terrorist Organizations

Sophisticated Hacktivists
Attack Targets – Digital Assets

- Emails
- Personal Data
- ... Anything Digitalized

Nations & Governments
IP Rich Organizations
Financial Services
Retailer and On-line Businesses
Critical Infrastructure Providers
Social Groups and Networks
Millions of Individuals at Once
Attacks on Software
Software is susceptible to different attacks

Different attacks need different protection
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 lighter obfuscation in the rest of the program
  - Faster generated code so that more security can be use 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 a new security paradigm well beyond traditional computer and network security.
Key Objectives of Software Protection

- Resist static and dynamic reverse-engineering
- Resist tampering (i.e. unauthorized modifications)
- Resist cloning (i.e. moving software to a node it is not authorized to run on)
- Resist spoofing (i.e. having software use false identification information, such as over a network)
- Hide both static and dynamic secrets, as they are created, moved, and used.
- Impede the production and distribution of useful “crack” programs.
- Facilitate timely, intelligent responses to crack incidents.
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

Source Code Protection:
- Code Transforms
- Asset / Key Hiding
- White-box Cryptography

Binary Protection:
- Integrity Verification
- Anti-Debug
- Secure Loader
- Secure Storage

Tools

Program Interlocking
- 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
- (Protected) Binary
- Binary Protection Tools
- Secured Libs & Agents
- Full Protected Binary

Native Execution Environment

Source Level Protection
Binary Level Protection
Unified Cloaking Toolset

- **C/C++ Source Code**
  - Transcoder Front-End
  - Cloaking Engine
  - Transcoder Back-End
  - Cloaked C/C++ Source Code
- **Source Code in Object-C, Java, #C and other languages**
  - LLVM Front-End Tools
  - F2L / L2F Converter
  - Binary Rewriting Tool
  - LLVM Optimizer & Other Tools
  - LLVM Compiler & Linker
- **JavaScript/Binary Code**
  - LLVM IR
  - Emscripten
  - Optimized & Cloaked asm.js, JS, Wasm
  - Optimized & Cloaked Native Code
Software Diversity: the-State-of-the-Art

- 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
Value of Software Protection

- **Secured Input**
  Authentication, validation, integrity, confidentiality of input data

- **Secured Output**
  Authentication, validation, integrity, confidentiality of output data

- **Hide Algorithms & Computations**

- **Tamper Resistance**
  Makes it hard to modify the code’s data and control flow

- **Hide Internal Data**
  Including internally initialized data

- **Tamper Detection**

- **Damage Mitigation**
# Power of Protection Technology (examples)

<table>
<thead>
<tr>
<th>Technology</th>
<th>Prevent Analysis</th>
<th>Prevent <em>Effective Tampering</em></th>
<th>Foil Automated Attacks</th>
<th>Supports Software Diversity</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>Program Interlock</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>

* Tampering which causes software to fail is less threat than software modified to achieve a hacker’s specific desired result
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
**Cloakware for Applications - Built on Core Technology**

## Cloakware for Applications

<table>
<thead>
<tr>
<th>Application Protections</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Protection</td>
</tr>
<tr>
<td>Node Locking</td>
</tr>
<tr>
<td>Jailbreak &amp; Root Detection</td>
</tr>
<tr>
<td>Diversity &amp; Renewability</td>
</tr>
<tr>
<td>Anti-Hooking</td>
</tr>
<tr>
<td>Java Access Control</td>
</tr>
<tr>
<td>Anti-Tamper</td>
</tr>
<tr>
<td>Anti-Reverse Engineering</td>
</tr>
</tbody>
</table>

### 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

### Cryptography
- White box Cryptography
  - RSA
  - AES
  - ECC
  - 3DES
- Whitebox PRNG
- Secure Store
- Secure Store

### Security Tools
- Secured Libraries (FlexLib)
- Smart Assembly
- Key Transformations
- Secure Code Injection
- Security Plugins
- Secure Heap
What can JavaScript and Webassembly Protection do?
Current Security Challenge of Web Application Environments

- Trusted
- Un-Trusted

Web Server

Client and Internet Environments

©2016 Irdeto, All Rights Reserved. – www.irdeto.com
Browser with asm.js Support and Wasm

As the next evolutionary step of asm.js, WebAssembly (Wasm) is a new project being worked by Mozilla, Microsoft, Google and Apple to create a new standard, that defines a portable, size- and load-time-efficient format and execution model specifically designed to serve as a compilation target for the web and non-web.
Is it possible to protect web applications running in a browser environment?

- It is Impossible!
- Don’t know how you can?

Rest of the world does not believe that this can be done by using software protection technologies.

Irdeto is a leader to develop new technology to protect and secure web applications by protecting JavaScript and Webassembly.
Security Capability of Irdeto JavaScript and Webassembly Protection

- Created a trusted digital platform for a protected web application inside web browsers
- Enforce integrity of the web application and protect 'business logic' running in web browsers
- Allow businesses to engage with their users in a more secure and reliable fashion to protect their business models

Now, a web application can be protected by itself even if in a hostile web browser
Overview of JavaScript/WASM Protection Technology

- **JS/WASM Cloaking Technology**
  New JS protection tool chain combines the Irdeto Transcoder with other enabling technologies such as LLVM and Asm.js

- **Direct JS Protection**
  New set of security protections applied directly to JS code analogous to Irdeto’s source and binary code protection features

- **New Trusted Platform for Tethered Web Applications**
  A new trust model leveraged JS/WASM cloaking and direct protection capabilities above
  - Server-based root-trust and security enforcements
  - Code and security behaviors: dynamic, randomized, agile, diversified and renewable during security life cycle
  - White-box encrypted messaging between client & server
New Trusted Digital Platform of Tethered Web Applications

Application Web Server

Server Security Elements and Manager
- Diversity and Renewability Manager
- Tamper Detection Manager
- JS Protection Tooling
- Shared-Secret Manager (Time Dependent Shared-Secrets)
- Secured Messaging Manager

Diversity and Renewability Manager

Shared-Secret Manager (Time Dependent Shared-Secrets)

Secured Messaging Manager

Client Browser Environment

DOM/Name Space
- $JS_{i1}$
- $JS_{i2}$
- $JS_{i3}$
- $JS_{ni}$

Client Security Elements and Protected App

Secured Local Storage (Shared-Secrets)

Application Web Server

©2016 Irdeto, All Rights Reserved. – www.irdeto.com
New Business Perspectives of Web Protection

New Products, New Markets, New Services, and New IP License Opportunities

Right Now
- Web Media Protection
- Online Advertising Anti-Fraud
- Secure Mobile and Online Payments
- Secure Mobile Game Billing

In the Future
- Protect Many Web Apps
  - IoT Security
  - Secure Virtual Client
  - HTML5 Offline Game Piracy
  - Online Banking
  - Online Gaming
  - Other Tethered Security
  - Secure Video and Voice Chat
  - Application SecureLets
  - Data SecureLets
  - Webmail Security

Building Irdeto Web Protection Technology and Solutions

JS/Wasm Cloaking - Direct JS Protection - Tethered Trust Model
Connected Application central based Security Model

Trusted model to address both man-in-the-middle and man-at-the-end attacks
Connected Application Topology

Monitoring/telemetry

Cloud

Updates

Web
desktop/laptop

Mobile

©2016 Irdeto, All Rights Reserved. – www.irdeto.com
Like the lifecycle for human health protection, the security lifecycle of a digital asset application mandates protection from creation, through distribution and then ultimately consumption from being deployed in the field.
Security Lifecycle Management

- Original Healthy Body & Improved Immunity
- Proactive Healthcare
- Monitoring & Diagnosis
- Medical and other Treatment
- Healing & Recovery

Healthcare is one of the largest industries and fields of research

- Software security is a young field
- Software protection is becoming main stream of system security
- Irdeto has best-in-class tools and technology for software security lifecycle

- Build in Attack Resistance and Robustness
- Proactive Security Countermeasures
- Monitoring and Analysis
- Attack Response
- Renew Security

Human Body Protection

Software Protection
Traditional Security Model

- Typical approach to security is to assume that the initial design will remain secure over time.
- Anything can be hacked given enough time and effort:
  - Set top boxes, PC apps, Mobile devices, CE devices
- 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
What is Dynamic Security?

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
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

Any software protection techniques can make own contributions to software diversity
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

Software Diversity Benefits

- Strong attack response
- Reduces duration of attack

Diverse production
- Raises cost of attack
- Reduces scope of attack

Resulting Hacker Business Model

Investment

Reward

Time

Secure Devices %

Response Breach

Time

Tamper resistance

0

100

90

Platform Compromised

Strong attack response

Reduces duration of attack

Diverse production

Diversity! Renewability!! Countermeasure!!
New View of Information Security and New Research Opportunity
New View of Information Security

- Security for Man-In-The-Middle Attack
- Security for Web Browser
- Security for Man-At-The-End Attacks

Static Security
Dynamic Security
Software Protection is largely different transformations with very different security profile comparing to traditional security.
Security vs. Practice

Technology Gap

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

Highly Practical

Much Less Practical

Impossible Program Obfuscation

Full Homomorphic Encryption

Indistinguishability Obfuscation

Cryptography and SW protection research can make good contributions
WB Security Problems

Problems Addressed by Homomorphic Transformation

- Homomorphic Encryption
- White-box Cryptography
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 has 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

- SW protection needs to prevent all attacks 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 the effectiveness of attacks by different skilled attackers?
- Security has to deal with unknown attacks in the future? How?
- Perfect security does not exist! SW security must be relative and renewable!

A good opportunity for research

Work with SW protection professionals to develop measurement model and metrics on SW security and protection (Good PhD research subjects)
Part 3: White-box Security Patterns
Introduction to 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
Advanced WBC Patterns (Covered by next generation of technology and products)

New WBC Patterns (for connected multiple environments)

Well Understood WBC Patterns (more about single environment)
Direct Attack Points (Just Examples)

Producing Software *(Trusted Environment)*

- Source code
- Compiling & Linking
- Executable

Distributing Software *(via 3rd Party Environment)*

- Build-in
- Pre-Install
- Executable

Running Software *(White-Box Computing Environment)*

- Data and data flow
- Operations
- Control flow
- Program decision and property
- Functions and invoking
- Executable
- Other Applications
- Run-time Environment

- Tampering Attacks
- Attack Tools
- Automatic Attack
- Cloning Attack
WB Security Pattern Coverage

Distributing Software (via 3rd Party Environment)

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

Pattern

- Secured loader
- Dynamic code decryption
- Homomorphic Data Transformation
- Control flow obfuscation
- Function Boundary Concealment
- Protecting program decision and property
- Software diversity
- Program integrity verification
- Anti debug
- Reinterpretation
- Execution flow integrity
- Sham detection
- White-box Cryptography
- SW anchoring

Tampering Attacks

- Attack Tools
- Automatic Attack

Operation

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

Run-time Environment

Running Software (White-Box Computing Environment)

Cloning Attacked

Executable

On-line
Download
### 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 Four 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

**Diagram:**
- **Main Memory**
  - Binary Code
  - Data
- **Secondary Storage**
- **CPU**
  - Fetch
  - Instruction Register
  - Registers
  - Execute (ALU)

**Attack Vectors:**
- White Box Attacks
  - Inspect and peruse data

**Legend:**
- White Box Attacks indicate security threats to data transformation.
Homomorphic Data Transformation: Security Problem

- 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
Homomorphic Data Transformation: Security Problem (2)

- 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.
Homomorphic Code/Data Transformation are Essential

- 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

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

Original data flow
Cloaked data flow
Relationship between smooth and transformed value

Transformed Space: \( Z' = F'(X', Y') \)

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.

Original X, Y and OP are disappeared.

New code for transformed computation (OP) and its result value (Z) in transformed form.
Homomorphic Data Transformation: in Memory and Operations

<table>
<thead>
<tr>
<th>Case</th>
<th>Memory</th>
<th>Register</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>null</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>null</td>
<td>null</td>
<td>T</td>
</tr>
<tr>
<td>4</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>5</td>
<td>T</td>
<td>T</td>
<td>null</td>
</tr>
<tr>
<td>6</td>
<td>T</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>7</td>
<td>T</td>
<td>null</td>
<td>T</td>
</tr>
<tr>
<td>8</td>
<td>null</td>
<td>T</td>
<td>null</td>
</tr>
</tbody>
</table>

Recommended use of transforms
Applicable cases
Avoided cases if possible

User can select transformations or preserving by using setting
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

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>z = x + y</td>
<td></td>
<td></td>
<td>z = 2x + y</td>
<td>z = x + 5</td>
</tr>
<tr>
<td>Transcoder transforms</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 + 10</td>
<td></td>
</tr>
<tr>
<td>z' = 5z + 17</td>
<td>z' = 10z + 4</td>
<td>z' = 5z + 3</td>
<td>z' = 5z -18</td>
<td></td>
</tr>
<tr>
<td>Transformed code segment</td>
<td>z' = x' + y'</td>
<td>z' = 2x' – y'</td>
<td>z' = 2x' – y'</td>
<td>z' = x'</td>
</tr>
</tbody>
</table>
WB 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
Execution Flow Integrity:
Security Context and Problem

- 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. a = 99
   c = E(c)
   case = b
   = 3

2. b = D(b)
   case = D(d)
   = 1

3. a = E(a)
   b = E(b)
   case = D(c)
   = 7

4. d = E(d)
   case = D(a)
   = 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 most fundamental technology adopted for traditional security problems.
White-Box Cryptography: Security Context – Cryptography Is Used Everywhere

- For applications that run in a hostile environment, cryptographic keys and other valuable assets become much easier and common attack targets for a multitude of purposes than in a trusted environment.

- In most business models, the recovery of some or all of these keys directly threatens the revenue from the applications, services, or digital assets.
White-Box Cryptography:
Security Context – Cryptographic Dilemma

- Cryptographic algorithms are well known to attackers because they are standards-compliant algorithms and well deployed by information systems (i.e. AES, RSA, ECC, SHA1).

- Whitebox context requires much more severer security challenges than traditional crypto attacks such as to black-box and side-channel attacks
  - any attack that can be mounted through the side-channel can be mounted more effectively via a direct channel
  - the information the side channel reveals can always be revealed through a direct channel as well
  - black-box crypto security does not work for white-box context
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:

  ```
  Four score and seven years ago our fathers brought forth on the continent a new nation conceived in Liberty and dedicated to the proposition that all men are created equal.
  ```

  Extract key from storage (encrypt, deobfuscate)

  ```
  Key
  ```

  Key is easily extracted by white-box attacker

  Data

  Decryption
White-Box Cryptography: Security Problem – White-Box Crypto Security

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!*

- 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?
White-Box Cryptography:
Security Intent

- The fundamental security intent of white-box cryptography is to make the recovery of the key in the whitebox 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 whitebox 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.
White-box cryptography ensures that input data, keys, intermediate results and output data are protected at all times by using homomorphic transformations.
White-Box Cryptography: Solution - White-Box Ciphers (examples)

- **AES and RSA algorithms**
  - AES-128bit / AES-256bit
  - RSA-1024bit to RSA-4096bit
  - WB-EC-DSA (sign, verify) standard NIST curves

- **Fixed Key WB-AES, WB-RSA and WB-EC-DSA**
  - Key is fixed and embedded in WBAES lookup tables

- **Dynamic Key WB-AES and WB-RSA**
  - Key is generated/supplied at runtime and transformed using data flow transformations

**Diagram:**

- **Fixed Key WB-AES, WB-RSA and WB-EC-DSA**
  - Cipher text $x^E_k$ to $D_k$ to Transformed Data $x^{T_1}$

- **Dynamic Key WB-AES & WB-RSA**
  - Key $k^{T_2}$ to Transformed Key
  - Cipher text $x^E_k$ to $D$ to Transformed Data $x^{T_1}$

- **White-Box Cryptography:** Solution - White-Box Ciphers (examples)
White-Box Cryptography: Solution - WB Implementation

- The key is mathematically inseparable from the surrounding data in which it’s been evaluated and embedded
  - Keeps a key hidden even if the attacker has visibility of the executing program
  - Increases the difficulty of key extraction
- The transformed key can be evaluated by an algorithm that may be different from the original cryptographic operation but that yields the same result as the published algorithm with the same input data
- WB ciphers can leverage data transformations to ensure that inputs to and outputs from white-box crypto operations do not appear in the clear
- Moreover, all transformed inputs, keys and outputs can be involved transformed computations before and after a white-box crypto operation.
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
The modification of application code and data is a common attack against application software.

The user and environment, which the application is running in, are untrustworthy.
- The user or environment could modify the application.
- The application cannot rely on its environment to report or protect against code and data tampering attacks.

There are a wide variety of freely available tools to allow an attacker to easily modify an application either statically or dynamically.

These tools may include hex editors, debuggers, disassemblers and tracers.
Static or dynamic code & data tampering can provide an attacker with the ability to modify the execution of the application resulting in

- An undesired behavior
- Escalating unauthorized privileges.
- Circumventing or breaking the copy protection on the application
Program Integrity Verification: Security Intent

- Program integrity verification is a kind of tampering resistant techniques to detect and react to tampering of an applications code and data
- Integrity verification of program image and data files on disk
  - Module level (executable, dynamic share libs, other binary and data files)
- Integrity verification of program binary in memory
  - Binary code
    - Module level (executable, dynamic share libs)
      - Single Module or multiple modules
    - Smaller fine grain level
      - a) Function level; b) Basic block level; c) Instruction level
  - Global constants
  - Export table of dynamic share libs
Program Integrity Verification: Solution – Signing Process at Build-time

- “On Disk” API function call verifies the entire file integrity
- “In Memory” API function call verifies portions of the code segment residing in memory
- Code Segment is partitioned into regions to speed up integrity checks
- Hash segments contain several interoperable regions
- Run-time Decryption for IV data uses White Box Crypto
Program Integrity Verification:
Solution – Self-hashing IV Technique

When, Where, and what for the IV check

Build-Time

Run-Time
Server based JS Integrity Verification

- The Integrity Verification is a server-side check which detects if the secure Javascript is being changed

1. Javascript source in the DOM or name space is hashed by the browser.
2. The hash value is sent to a server component
3. The server checks that the hash value matches the expected value for the specific Javascript instance

- Each Javascript instance uses a different key for the HMAC
- IV is a challenge-response mechanism. Data from the server provides randomness to the hash values calculated to prevent replay attacks
Integrity Verification

Client
- Application
- Irdeto JS

Irdeto Services
- Diversification
- Verification

- load Javascript instance
- init
- random input to hash
- calculate hash of Javascript source
- payment message including hash value
- get Javascript instance sent to the client
- check expiry of Javascript
- calculate correct hash
- compare against value in message
Runtime Verification

Virtual Private Cloud (VPC)

Diversifier

workers

virtual private cloud

Protected JavaScript

Cloudfront distribution

Different JavaScript per client

Self-checks

Application Server

Message body + custom header

Verifier

virtual private cloud

Message body + self-check data
Summary

- Black-box and grey-box security models are inadequate for many important software applications
- We need more research into creating software that is secure in the white-box attack model
- White-Box attacks are much more difficult security problems
- White-Box security is a new challenge for both industrial and academic communities
- Software security needs software protection solutions and methods across the security lifecycle
- Software protection is a very young field and many open problems are new opportunity for talent students and researchers to resolve
- Irdeto is a leader in digital asset protection technology with considerable uptake worldwide
- Research collaboration and internship with Irdeto are encouraged
Question?

Thanks!