Advanced Software Protection: Integration, Research, Exploitation

Bjorn De Sutter
bjorn.desutter@ugent.be
Aspire in a nutshell

2

SafeNet use case → Software Protection Tool Flow → Protected SafeNet use case

Gemalto use case → Software Protection Tool Flow → Protected Gemalto use case

Nagravision use case → Software Protection Tool Flow → Protected Nagravision use case

Data Hiding | Algorithm Hiding | Anti-Tampering | Remote Attestation | Renewability

Aspire: Advanced Software Protection: Integration, Research and Exploitation
Man-At-The-End (MATE) Attacks
Man-At-The-End (MATE) Attacks

Software analysis tools
oscilloscope

developer boards
screwdriver
JTAG debugger

Aspire: Advanced Software Protection: Integration, Research and Exploitation
Economics of MATE attacks

Engineering a.k.a. identification

Exploitation

€/day

Protection

Time
Economics of MATE attacks

€/day

engineering
a.k.a. identification

protection

exploitation

diversity

time
Economics of MATE attacks

- Engineering a.k.a. identification
- Protection
- Exploitation
- Diversity
- Renewability

€/day

Time
<table>
<thead>
<tr>
<th>Asset category</th>
<th>Security Requirements</th>
<th>Examples of threats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private data (keys, credentials, tokens, private info)</td>
<td>Confidentiality, Privacy, Integrity</td>
<td>Impersonation, illegitimate authorization, Leaking sensitive data, Forging licenses</td>
</tr>
<tr>
<td>Public data (keys, service info)</td>
<td>Integrity</td>
<td>Forging licenses</td>
</tr>
<tr>
<td>Unique data (tokens, keys, used IDs)</td>
<td>Confidentiality, Integrity</td>
<td>Impersonation, Service disruption, illegitimate access</td>
</tr>
<tr>
<td>Global data (crypto &amp; app bootstrap keys)</td>
<td>Confidentiality, Integrity</td>
<td>Build emulators, Circumvent authentication verification</td>
</tr>
<tr>
<td>Traceable data/code (Watermarks, finger-prints, traceable keys)</td>
<td>Non-repudiation</td>
<td>Make identification impossible</td>
</tr>
<tr>
<td>Code (algorithms, protocols, security libs)</td>
<td>Confidentiality</td>
<td>Reverse engineering</td>
</tr>
<tr>
<td>Application execution (license checks &amp; limitations, authentication &amp; integrity verification, protocols)</td>
<td>Execution correctness, Integrity</td>
<td>Circumvent security features (DRM), Out-of-context use, violating license terms</td>
</tr>
</tbody>
</table>
Aspire in a nutshell

Data Hiding  Algorithm Hiding  Anti-Tampering  Remote Attestation  Renewability

ASPIRE Framework
Decision Support System
Software Protection Tool Chain

Protected SafeNet use case
Protected Gemalto use case
Protected Nagravision use case

SafeNet use case
Gemalto use case
Nagravision use case

Aspire: Advanced Software Protection: Integration, Research and Exploitation
1. Reference architecture for protected mobile services

- Mobile device (untrusted, MATE attack)
  - Client-side app: hidden data, hidden algorithms, anti-tampering mechanisms
  - Renewability-supporting virtual machine
  - Remote attester

- Wireless/mobile network (untrusted, MITM attack)
  - Secure channel
  - ASPIRE protected program

- Server (trusted)
  - Server-side logic: remote verifier, bytecode provider, renewalability protection engine

2. Software protection techniques and integrated plugin-based tool flow

- Annotated source code
  - ASPIRE source level protection: data hiding, algorithm hiding, anti-tampering
  - Partially protected source code
  - Standard compiler
  - Object code

- ASPIRE protected program
  - Client-side app
  - Server-side logic

- ASPIRE binary level protection: data hiding, algorithm hiding, anti-tampering, security libraries
  - Remote attestation
  - Renewability
3. Decision Support System

- attack models & evaluation methodology
- security metrics
- experiments with human subjects
- public challenge

2. Software protection techniques and integrated plugin-based tool flow
Part 1: Reference Architecture

- Cookbook for combining protections
- Why?

![Diagram of Petri Net]

- Start of the attack
- Attack steps
- Sub-goal
- Final goal

Aspire: Advanced Software Protection: Integration, Research and Exploitation
Part 1: Reference Architecture

- How to combine multiple protections?
  - How do the individual protections actually work?
Part 1: Reference Architecture

- How to combine multiple protections?
  - How do the individual protections actually work?

```
1. The original application transfers control to the stub.
   Details: Currently this is implemented as an unconditional jump into the first part of the stub code. Conceptually but not yet implemented this jump could be removed by Diablo by means of branch forwarding, so that the stub is inlined in the application code.

2. The stub sets up state for VM and transfers control.
   Details: The stub collects the contents of the physical ARM processor registers and calls the VM, passing the address of the corresponding bytecode (VM-image) as argument. When different stubs have different entry points into the VM, those entry points can be inlined in the stubs as well.

3. The VM fetches the Bytecode and interprets it.
   Details: In case the bytecode is stored in encrypted form, the VM will need to decrypt it during this process.

4. After interpretation is finished, control is transferred to second part of the stub.
   Details: The bytecode comprises code to calculate the address where the native execution should continue. This address and the updated register values are returned to the stub.

5. The stub cleans up and transfers control back to the application.
   Details: The stub updates the physical ARM registers with the values the VM returned and jumps to the continuation address, transferring control back to the application.
```
Part 1: Reference Architecture

- How to combine multiple protections?
  - How do the individual protections actually work?
Part 1: Reference Architecture

- How to combine multiple protections?
  - How do the individual protections actually work?

  - data obfuscations
  - white box cryptography (static keys, dynamic keys, time-limited)
  - diversified crypto libraries
Part 1: Reference Architecture

How to combine multiple protections?
- How do the individual protections actually work?

- control flow obfuscations
- multithreaded crypto
- instruction set virtualization
- code mobility
- self-debugging
- client-server code splitting
Part 1: Reference Architecture

- How to combine multiple protections?
  - How do the individual protections actually work?

- code guards
- static and dynamic remote attestation
- reaction mechanisms
- client-server code splitting
Part 1: Reference Architecture

- How to combine multiple protections?
  - How do the individual protections actually work?

- Data Hiding
- Algorithm Hiding
- Anti-Tampering
- Remote Attestation
- Renewability

- native code diversification
- bytecode diversification
- renewable white-box crypto
- mobile code diversification
- renewable remote attestation
Part 1: Reference Architecture

- How to combine multiple protections?
  - How do the individual protections actually work?
  - How do the protections compose?
  - Do the protections share components?
  - If protections compose, are there phase-ordering issues?
  - Which protections/components need to be combined and how?
  - Where is $1 + 1 > 2$ in terms of protection strength?
  - What is the combined impact on software development life cycle?
2. Software protection techniques and integrated **plugin-based** tool flow

- **Python** – Dolt compiler flow
- **JSON** configuration scripts
- invokess chain of +/- independent tools
- **TXL source code rewriting**
- **Diablo link-time binary rewriting**
static const char ciphertext[] __attribute__ ((ASPIRE("protection(wbc,label(ExampleFixed),role(input),size(16))")))
  = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
      0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f };

static const char key[] __attribute__ ((ASPIRE("protection(wbc,label(ExampleFixed),role(key),size(16))")))
  = { 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
      0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xe, 0xff };

char plaintext[16] __attribute__ ((ASPIRE("protection(wbc,label(ExampleFixed),role(output),size(16))")))
  ;

Pragma("ASPIRE begin protection(wbc,label(ExampleFixed),algorithm(aes),mode( ECB),operation(decrypt))")
decrypt_aes_128(ciphertext, plaintext, key);
Pragma("ASPIRE end");
Source Code rewriting

A data flow analysis is needed to decide where to insert these operations, and also to check whether it is possible at all to apply a data hiding protection to a variable: when alias analysis cannot guarantee that a variable will only be accessed by a limited set of read and write operations that can all be rewritten, it is not safe to apply a transformation. GrammaTech's CodeSurfer provides a framework on top of which custom data flow analyses can be implemented. For the data hiding protections, and later also for the client-server code splitting, custom analyses will be developed that compute the required data flow information in SLP05.01.

The data hiding source code rewriter will then rewrite the software as requested by the source code annotations (see Section A) and to the extent allowed by the data flow analysis results. In later versions of the tool flow, the analyses will be extended to analyze to what extent data hiding annotations can be propagated throughout the program.

For a simple example, suppose there is a function

\[
\text{int add(int x, int y)}
\]

that simply adds the values of \(x\) and \(y\) and returns the sum. Then consider a code fragment

\[
\text{int x __attribute__((ASPIRE("protection(xor,mask(constant(12)))"))) a = 5;}
\]
\[
\text{int x __attribute__((ASPIRE("protection(xor,mask(constant(12)))"))) b = 6;}
\]
\[
\text{int x __attribute__((ASPIRE("protection(xor,mask(constant(12)))"))) c = add(a,b);}
\]

Ideally, this fragment should not be rewritten into

\[
\text{int a = 5}^{12};
\]
\[
\text{int b = 6}^{12};
\]
\[
\text{int c = add(a}^{12},b^{12})^{12};
\]

but instead it should become something along the following lines:

ASPIRE D5.01 CONFIDENTIAL 25
Client-side code splitting (D1.04 Section 3.1) is one of the protections that will be implemented in the three first steps as discussed in Section 9.1. In its initial implementation, a fixed SoftVM, which requires no customization, will be embedded in the code to protect.

### 9.3.1 BLP01: Native Code Extraction

As indicated in Figure 13, in BLP01.01 a Diablo rewriter will collect the code fragments that need to be translated from native code to bytecode. It does so on the basis of the annotation facts D01 as assembled by the source-level component SLP04, and based on its usual inputs, which in this case correspond to the application BC02 to be rewritten, the corresponding map file (D02) and the object code (BC08) that was linked into the original application by the standard linker.

**Figure 13: Tool flow components for chunk extraction and bytecode generation**

Diablo produces a description of the native code chunks in the form of JSON files (BLC02). The specification for this interface is presented in Appendix D.

To select the native code fragments to be translated to bytecode, the Diablo tool will consider procedures marked as such in the annotation facts D01. Within these fragments, all possible fragments will be selected, i.e., all fragments of which the instruction selector indicates that the instructions in them are supported by the X-translator and the SoftVM.

### 9.3.2 BLP02: Bytecode Generation

The second tool BLP02 in support of client-side code splitting is the X-translator. Based on the JSON files of BLC02 it generates bytecode, as well as stubs that will replace the selected native code fragments. The responsibility of the stub is to invoke the SoftVM that will be embedded in ASPIRE.
Part 3: Decision Support

- Knowledge Base
- Complexity & Resilience Metrics
- Protection Strength Evaluation Methodology
- Optimization strategies

Aspire: Advanced Software Protection: Integration, Research and Exploitation
Validation & Demonstration

- three real-world use cases
  - software license manager
  - one-time password generator
  - DRM protection

- security requirements from industry
  - functional requirements
  - non-functional requirements
  - assurance requirements

- dynamically linked Android 4.4 – ARMv7 libraries

- penetration tests by professional pen testers
Validation & Demonstration

- controlled experiments with academic hackers
- public challenge and bounties
More resources

- [https://www.aspire-fp7.eu](https://www.aspire-fp7.eu)
  - papers
  - public reports
  - contact info

- [https://github.com/aspire-fp7](https://github.com/aspire-fp7)
- [https://github.com/diablo-rewriter](https://github.com/diablo-rewriter)

- Youtube channel: ASPIRE-FP7 Software Protection Demonstration
The Aspire project has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement number 609734.

If you need further information, please contact the coordinator:
Bjorn De Sutter, Ghent University
Technologiepark-Zwijnaarde 15, B-9052 Gent, Belgium
Tel: +32 9 264 33 67  Fax: +32 9 264 35 94
Email: coordinator@aspire-fp7.eu
Website: https://www.aspire-fp7.eu

The information in this document is provided “as is”, and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.