OMTP PUBLISHED

OMTP

SECURITY THREATS ON EMBEDDED CONSUMER DEVICES

VERSION:	v1.1
STATUS:	Approved for Publication
DATE OF PUBLICATION:	28th May 2009
OWNER:	OMTP Limited

CONTENTS

The information contained in this document represents the current view held by OMTP Limited on the issues discussed as of the date of publication.

This document is provided “as is” with no warranties whatsoever including any warranty of merchantability, non-infringement, or fitness for any particular purpose. All liability (including liability for infringement of any property rights) relating to the use of information in this document is disclaimed. No license, express or implied, to any intellectual property rights are granted herein.

This document is distributed for informational purposes only and is subject to change without notice. Readers should not design products based solely on this document.

Each Open Mobile Terminal Platform member and participant has agreed to use reasonable endeavours to inform the Open Mobile Terminal Platform in a timely manner of Essential IPR as it becomes aware that the Essential IPR is related to the prepared or published specification. The declared Essential IPR is publicly available to members and participants of the Open Mobile Terminal Platform and may be found on the “OMTP IPR Declarations” list at the OMTP Members Access Area.

The Open Mobile Terminal Platform has not conducted an independent IPR review of this document and the information contained herein, and makes no representations or warranties regarding third party IPR, including without limitation patents, copyrights or trade secret rights. This document may contain inventions for which you must obtain licenses from third parties before making, using or selling the inventions.

Defined terms and applicable rules above are set forth in the Schedule to the Open Mobile Terminal Platform Member and Participation Annex Form.

1 INTRODUCTION

1.1 DOCUMENT PURPOSE

When defining the Security Policy of a User Equipment (UE, as defined in 3GPP TR 21.905 [1]), the relevance of each of the Threats applicable to UE Assets should be taken into consideration. Indeed, protecting specific Assets against specific Threats must be considered as a balance of cost of security protection vs. cost of security break.

Cost of security protection encompasses:

- Increased development time
- Increased silicon cost
- Increased manufacturing and test time, including:
    - Installing the security
    - Verifying the security
- Increased device complexity
- Decreased device performance

Cost of security break includes:

- Potential cost of the device
- Service and support in countering the break
- The cost can be in reputation or pure financial terms and can affect
    - The manufacturer
    - The software provider
    - The information owner
    - The information user
    - The operator

This document defines a list of common Threats that may apply to a UE. The potential danger of each Threat, as well as the ease of creation, spreading, and blocking of each Threat is also described.

1.2 INTENDED AUDIENCE

This document is written for those who have an understanding of security and require details of possible Threats applicable to UEs.

1.3 REQUIRED EXPERTISE

The level of required expertise reflects the expertise of a Threat Agent and the availability of tools required to find a vulnerability that can be exploited.

Easy	Can be implemented by a UE user
Moderate	Requires expert knowledge to perform
Hard	Requires expert knowledge and expensive laboratory resources to perform

1.4 EASE OF REPEAT

The ease of repeat reflects the expertise, cost and time to reproduce the Attack on UEs other than the device where the vulnerability was originally found.

Easy	Can be repeated by UE user
Moderate	Can be repeated by backstreet shop
Hard	Requires expert knowledge and expensive laboratory resources to repeat

1.5 EASE OF DISTRIBUTION

The ease of distribution reflects how easy it is to spread the result of the Attack, or to distribute methodology in order to reproduce the Attack.

WWW	Information can be spread on the World Wide Web once the first break is accomplished, typically distributed as software to repeat the exploit or as the discovered secret. This can be considered indicative of the worst case.
Small Corporate	Requires a small corporation to make use of information / distribute the security crack, e.g. to manufacture modification chips, called hereafter "Mod Chips"
Corporate	Requires a large corporation to make use of the information, e.g. to create and sell cloned phones

Some Attacks directly create a result, such as running a piece of software that allows a device to ignore DRM when played on that device.

Some Attacks result in a piece of data that can then be used elsewhere. For example, a piece of software that runs on a device and sends the owner's credit card details (the result) back to a thief for future use.

Any Attack that reveals a class secret (such as the credit card details above) is immediately considered "WWW" distributable (synonymous with the worst case).

1.6 THREATS CLASSIFICATION

While it is possible to divide Threats into a number of classifications and to varying degrees of granularity, this document divides Threats into six categories to enable intuitive understanding of the Threat (with associated Required Expertise, Ease of Repeat and Ease of Distribution) as well as to keep the number of categories to a minimum in order to maintain simplicity within the document.

The six categories of Threats and their definitions are listed below

1. Software Modification Threats (T.SWM.xxx)
   - Logical Threats aiming to modify the software of the UE.
2. Software Opportunistic Threats (T.SWO.xxx)
   - Logical Threats aiming to take advantage of a weakness in either the definition or implementation of the software in the UE. The Threat could expose secrets or cause the terminal to behave in an unintended or unauthorised way.
3. Hardware Threats – External (T.HWE.xxx)
   - Physical Threats which can be implemented without any breach of the terminal's Integrity, generally through the ports and connectors available outside of the UE.
4. Hardware Threats – Terminal Intrusive (T.HWT.xxx)
   - Physical Threats which are implemented by opening the outer encasing of the UE. These include probing of busses on the PCB or the exposed pins of a package mounted on the PCB. This includes the physical removal of a component for offline Attacks if the component's Integrity is not physically damaged in the removal process or the Attack.
5. Hardware Threats – Component Invasive (T.HWC.xxx)
   - Physical Threats or Attacks which are implemented by affecting the physical Integrity of the component (breach or destruction), including but not limited to the SoC, Memories, and PCB. This includes the physical removal of a component for offline Attacks if the component's

Integrity is physically damaged in the removal process or the Attack.

6. Hardware Cloning, Component Replacement or Component Addition Threats (T.CLO.xxx)
    - Physical Threats which consist of replacing a part or the entire UE with an alternative component, or of adding a component to the UE. This includes but is not limited to replacing the SoC or Memory with another SoC or Memory as well as creating a copy of the UE.

2 Threats Summary

What follows is a brief overview of the Threats which may be considered. A more detailed explanation of each Threat can be found in section 4.

2.1 Software Modification Threats (T.SWM.xxx)

Threat Description	Required Expertise	Ease of Repeat	Ease of Distribution
T.SWM.001 Attack via faulty privileged code extensions (e.g. drivers)	Moderate	Easy	WWW
T.SWM.002 Attack via illegal privileged code extensions (drivers)	Moderate	Easy	WWW
T.SWM.003 Unauthorised re-flash of device through FOTA	Hard	Hard	Small Corp
T.SWM.004 Subverting of general software loading procedures	Moderate	Moderate	WWW

2.2 Software Opportunistic Threats (T.SWO.xxx)

Threat Description	Required Expertise	Ease of Repeat	Ease of Distribution
T.SWO.001 Attack via faulty OS code(bug)	Moderate	Easy	WWW
T.SWO.002 Attack, via unrelated application APIs, to secure resources	Moderate	Easy	WWW
T.SWO.003 Breaking Access Control performed by software to hardware features	Moderate	Easy	WWW
T.SWO.004 Breaking Access Control performed by software to confidential data, code and keys	Moderate	Easy	WWW
T.SWO.005 Buffer overflows	Moderate	Easy	WWW

THREAT DESCRIPTION	**REQUIRED EXPERTISE**	**EASE OF REPEAT**	**EASE OF DISTRIBUTION**
T.SWO.006 Code verifiability related security holes	Moderate	Easy	WWW
T.SWO.007 Unpredictable CPU instructions	Hard	Easy	WWW
T.SWO.008 DMA or CLCD use for accessing memories	Moderate	Easy	WWW
T.SWO.009 Faking of general software identity	Moderate	Easy	WWW
T.SWO.010 CLCD use for displaying memories and interfering with displayed data	Moderate	Easy	WWW
T.SWO.011 Attack through uncontrolled API in general software space	Moderate	Easy	WWW
T.SWO.012 Attack through uncontrolled instruction set space	Hard	Easy	WWW
T.SWO.013 Attack through interaction of software concurrent processes causing logical breaks	Hard	Moderate	WWW
T.SWO.014 Exploit software bugs in execution environment	Moderate	Easy	WWW
T.SWO.015 Software Attack on Type Unsafe APIs inside the execution environment	Moderate	Easy	WWW
T.SWO.016 Software Attack on Type-Safe APIs inside the execution environment	Hard	Easy	WWW
T.SWO.017 Attack Through virtual debug port	Moderate	Easy	WWW

2.3 Hardware Threats – External (T.HWE.xxx)

Threat Description	Required Expertise	Ease of Repeat	Ease of Distribution
T.HWE.001 Unauthorised access via external invasive or non-invasive debug ports (e.g. JTAG, ETM)	Moderate	Moderate	Small Corp
T.HWE.002 Unauthorised re-flash of device through external debug port (e.g. JTAG)	Moderate	Moderate	Small Corp
T.HWE.003 Unauthorised re-flash of device through external serial interface	Moderate	Easy	Small Corp
T.HWE.004 Bypass security by external battery removal	Easy	Easy	WWW
T.HWE.005 Bypass security by external memory card removal	Easy	Easy	WWW
T.HWE.006 Scan chain attack (direct or side channel)	Moderate	Moderate	Small Corp
T.HWE.007 Built-in self-test	Moderate	Moderate	Small Corp

2.4 Hardware Threats – Terminal Intrusive (T.HWT.xxx)

Threat Description	Required Expertise	Ease of Repeat	Ease of Distribution
T.HWT.001 Extract secret via Bus monitoring (hardware probes)	Moderate	Moderate	Small Corp
T.HWT.002 Unauthorised access via internal but off-SoC invasive or non-invasive debug ports (e.g. JTAG, ETM)	Moderate	Moderate	Small Corp
T.HWT.003 General hardware Attack on data in external RAM (e.g. probing)	Hard	Moderate	Small Corp

THREAT DESCRIPTION	REQUIRED EXPERTISE	EASE OF REPEAT	EASE OF DISTRIBUTION
T.HWT.004 Hardware Attacks on dynamic information in internal RAM (on-SoC, outside IC package)	Hard	Hard	Small Corp
T.HWT.005 Power analysis Attack to reveal secrets	Hard	Hard	WWW
T.HWT.006 Time analysis Attack to reveal secrets	Hard	Hard	WWW
T.HWT.007 Bypass security by glitch Attacks (e.g. power)	Hard	Hard	WWW
T.HWT.008 Bypass security by power removal to NV memory	Moderate	Moderate	WWW
T.HWT.009 Attack through interaction of hardware concurrent processes causing logical breaks	Hard	Moderate	WWW

2.5 HARDWARE THREATS – COMPONENT INVASIVE (T.HWC.XXX)

THREAT DESCRIPTION	REQUIRED EXPERTISE	EASE OF REPEAT	EASE OF DISTRIBUTION
T.HWC.001 Extract secret via Bus monitoring (de-cap/drill & hardware probes)	Hard	Hard	Small Corp
T.HWC.002 Hardware Attacks on static information in internal RAM (on-SoC, inside IC package)	Hard	Hard	Small Corp
T.HWC.003 Hardware Attacks on dynamic information in internal RAM (on-SoC, inside IC package)	Hard	Hard	Small Corp
T.HWC.004 De-capping of the chip holding secrets	Hard	Hard	Small Corp

THREAT DESCRIPTION	REQUIRED EXPERTISE	EASE OF REPEAT	EASE OF DISTRIBUTION
T.HWC.005 Focused Ion Beam (FIB) manipulation	Hard	Hard	Small Corp
T.HWC.006 Probe stations	Hard	Hard	Small Corp

2.6 HARDWARE CLONING, COMPONENT REPLACEMENT OR COMPONENT ADDITION THREATS (T.CLO.XXX)

THREAT DESCRIPTION	REQUIRED EXPERTISE	EASE OF REPEAT	EASE OF DISTRIBUTION
T.CLO.001 Cloning device by copying PCB and Flash	Hard	Hard	Corporate
T.CLO.002 External RAM Chip Replacement Attack	Hard	Moderate	Small Corp
T.CLO.003 Hardware Attacks to change static information in external RAM	Hard	Moderate	Small Corp
T.CLO.004 Hardware Attacks to change dynamic information in external RAM	Hard	Moderate	Small Corp
T.CLO.005 Attack by replacement of Flash when power is off (pre-boot)	Hard	Moderate	Small Corp
T.CLO.006 Attack by replacement of Flash when power is on (post-boot)	Hard	Moderate	Small Corp

3 TOOLS IN THE ARMOURY TO STOP THE HACKER

The information that follows covers a number of areas that should be considered in any security conscious system design.

3.1 PRIVILEGED CODE AND ITS ROLE IN GUARDING SECRETS

Many Threat descriptions contain a reference to unauthorised code privilege escalation into kernel mode.

Operating systems typically use a memory management unit (MMU) in the CPU to enforce code and data isolation and hence provide security to a set of features in the operating system. This isolation is controlled by code running in privileged mode.

Unfortunately, as well as the MMU control code, other code also runs in privileged mode and there are typically many ways of entering privileged mode. Once any rogue code is able to execute in the privileged state it is much easier for it to extract secrets or modify any security functionality in a system. Therefore, breaking into privileged mode is a key initial goal of many security cracks.

3.2 APIS AND SECURITY

Code uses APIs to communicate between one task and another. Many Attacks are focused on misuse of APIs, either to cause the APIs to provide information they are not meant to expose, or to cause the code on the other side of the API to perform illegal actions.

In general, APIs are another key area of Attack, and are one of the routes to break into privileged mode.

One reason that APIs are often vulnerable is that they are designed with goals other than security in mind. Typically speed and functionality are the primary drivers for general API development.

A typical OS may have hundreds or thousands of APIs across the user/privileged boundary, many of which cannot be 'secured' as this will break legacy code.

3.3 TYPE-SAFETY

Type describes the sort of data a block of memory holds.

This description may specify the following:

- That a particular data element is a string or an integer or a floating point etc.
- How many bytes of memory it resides in
- The value range it may occupy (1 to 5, Monday to Friday etc)

Fundamentally, data in a Type-Safe API is strongly checked to make sure it is what it claims to be, and often contains information related to more than one of the above 3 points attached directly to each block of data.

Typical APIs are not written to be Type Safe, as there is an overhead to be paid in terms of performance and flexibility to perform this checking.

If an API set between user and privileged space is Type-Safe then that gives strong assurance that that interface will not leak data through API misuse (it may still leak through logic flaws in the code behind the API). However, if there is both a Type-Safe API between user and privileged mode, and a non-Type-Safe API covering a different set of functionality, then the Type-Safe API is vulnerable to flaws in the non-Type-Safe API (e.g. a buffer Overflow Attack breaking the privileged /user boundary).

3.4 SECURITY DOMAINS

Security domains are areas of code or memory, either logically or physically separate from the general software domain of a general OS. These area(s) contain code and data of specific security interest.

Unfortunately, the term domain is used for a number of purposes – even within the security industry. To prevent confusion, this document refers to the secure world and normal (or non-secure) world, rather than secure or non-secure domains.

3.5 SECURITY CRITICAL ASSETS VS NON-CRITICAL ASSETS

Throughout this document, a general OS is discussed as a source of many potential Threats. This is because in a typical embedded environment there are two goals from the platform point of view:

1. Provide a rich user experience with versatile connectivity to different external information sources
2. Keep secrets on the platform safe from extraction

Separation of these two goals has advantages, at least in clarity of objectives. Take as an example a device with secure connection over TCP/IP:

- On the device there may be a TCP/IP stack to provide Internet connectivity
- It may also require HTTPS to establish a secure connection
- HTTPS sits on top of TCP/IP
- TCP/IP internals do not have to be secure, as those same internals will be duplicated on numerous routing servers between the device and, for example, a bank
- Security software only has to worry about placing the HTTPS protocol on top of TCP/IP, and would actually be harmed by the inclusion of the TCP/IP stack in its 'domain' due to the dangers

of code bloat and the need for verification of this code before release

This leads to considering a device as containing a general OS with the entire rich user interface, peripheral connectivity and protocol provision services, and a separate domain containing the security critical code and data segments.

Why consider this separation?

- The number of potential flaws in security can be directly linked to code size
- One boundary used for two tasks, security and "other operations", is flawed from the security point of view by exploitable flaws in those "other operations"
- Change of code behind a security boundary should be kept to a minimum. It should be separable from a general system re-build

If security tasks can be separated from general tasks then the critical code size and other vulnerabilities can be reduced, and as a bonus extra precautions can be taken that couldn’t be applied to the general code (e.g. relocate code to a physically safer space)

3.6 INTEGRITY CHECKING

One solution to the problem of "hacks" modifying a system is to perform Integrity checking on the current system code and data.

This requires three things:

- knowledge of the correct state of that system code and data;
- a trustable entity that verifies that the system code and data is in a correct state”;
- an effective and non-circumventable response to an integrity failure.

One way to perform integrity checking is to calculate a signature value over a block of data and compare that value to a reference value. Data and code can be checked by software, running in a more trusted space, or by hardware. The time to perform those checks can be considerable.

For instance:

A typical hashing algorithm takes around 22 Cycles per byte.

On a 200MHz CPU that gives

0.22 Seconds to check an entire 2Mb RTOS

2.2 Seconds to check an entire 20Mb smartphone OS

This speed restriction may raise problems in a system design.

Integrity checking works best where information is in a relatively static environment.

If some other task is moving the information around then the Integrity checker has to be informed (and trust that other task to be doing legal movements).

If the data is dynamic then the Integrity checker has to modify its reference values each time that data is changed (and trust the changer was performing a legal action).

Suggested Solutions:

- Consider how often such a check must occur:
    - For Integrity failures that lead to instantaneous loss of a valuable resource, any instance of a security break is critical
    - For Integrity failures that have long term consequences on a device, (e.g. IMEI modification on phones) then it may be acceptable to test the Integrity over a period of days or weeks.
    - If the memory cannot be attacked after it is loaded, then that validation check only has to be done on load.
        - This is why Secure Boot is so critical.
- Consider if everything needs to be checked:
    - If software is placed behind a security boundary that is safe from all considered Threats, then it does not need to be checked once it has been put in place
    - Security software may require a TCP/IP stack to transmit its secrets to remote devices – but that security software will be running safeguards on top of the TCP/IP to prevent interception of data (e.g. HTTPS), so the TCP/IP stack itself may not have to be checked for breaks
    - Similar arguments can be considered for much of the graphical and I/O environment of a device
    - In a typical general OS, the kernel and Access Control software are only a few hundred kilobytes. If that is verified then that may be sufficient re-assurance
- Consider who does the checking
    - If Integrity is validated in one part of a device, then that part can act as the gatekeeper to other functionality
    - Such Integrity can be built in a tree of trust, with one part offering validation to another, as long as the susceptibility

to Attack of that one part is considered lower than that which it is validating

3.7 SECURE BOOT

Trust in a system is built up from a well known defined point, known as the "root of trust". Typically, this might be an on-chip key, which can be used to verify code and data brought in from elsewhere in the system. Where that root of trust is used to validate the operating environment for the Execution Environment, the process is known as Secure Boot, and establishes that:

- The system booted a valid OS or Virtual Machine Monitor (VMM)
- The system loaded valid drivers and kernel modules
- The system loaded valid applications

By "valid" it means that the code and static data being loaded is that which is expected by the secure boot authority.

If the Secure Boot process completes correctly, there is an assurance that the booted execution environment is one that the Secure Boot authority (e.g. device manufacturer or network operator) approved.

4 THREATS DETAILS

The following list of Threats is not comprehensive but should act as a starting point when considering a system design.

Each Threat comes with a description of the Threat and a list of suggested solutions that may or may not be appropriate on a particular device.

4.1 SOFTWARE MODIFICATION THREATS (T.SWM.XXX)

<table>
<tr><th>THREAT DESCRIPTION</th><th>REQUIRED EXPERTISE</th><th>EASE OF REPEAT</th><th>EASE OF DISTRIBUTION</th></tr>
<tr><td><b>T.SWM.001</b> Attack via faulty privileged code extensions (e.g. drivers)</td><td>Moderate</td><td>Easy</td><td>WWW</td></tr>
<tr><td colspan="4">Description:
This form of Attack occurs when a typical OS is forced to allow changes to privileged code due to the need to enable changing of driver software. Driver software in such an OS typically has privileged access to handle interrupts from peripherals.
One problem in this area is that drivers are typically written by third parties and drivers are upgraded at different times to the main OS. This causes problems when implementing any "whole device" validation scheme, or even isolated checking of specific software updates
This code has a relatively high turnover for privileged code, is not written with a security focus, and therefore is vulnerable to a higher incidence of security flaws.
Suggested Solutions:
• Prevent downloading of privileged code
• Introduce a scheme of privileged code signing and certification
• Reduce size of privileged code using VMM or user-mode drivers techniques
• Introduce another level of security safe from privileged access</td></tr>
<tr><td><b>T.SWM.002</b> Attack via illegal privileged code extensions (drivers)</td><td>Moderate</td><td>Easy</td><td>WWW</td></tr>
<tr><td colspan="4">Description:
This form of Attack occurs because a typical OS is forced to allow changes to Privileged code through changes to driver software. Driver software in such an OS has privileged access if it is not running on top of a VMM.
Threat Agents intentionally write software to appear as a drivers to the OS,</td></tr>
</table>

thus enabling the exploitation of this access loophole.

Suggested Solutions:

- Prevent unauthorised downloading of privileged code
  - Only permit download from trusted sources
  - Only permit download when activated by an authorised route
- Prevent downloading of privileged code
- Introduce a scheme of privileged code signing and certification
- Reduce size of privileged code using VMM, user-mode drivers techniques
- Introduce another level of security safe from privileged access

T.SWM.003 Unauthorised re-flash of device through FOTA	Hard	Hard	Small Corp

Description:

FOTA (Firmware Over The Air) has to be considered separately from the general execution environment of a device, as it often works as part of the boot sequence, replacing one version of the general OS(s) with another.

The Threat is in the use of this valid update technique with invalid data.

This is the sort of Attack that may occur as soon the current range of more direct re-flashing Attacks is stopped. It requires the update server's message to be faked at some point in its life cycle, or to place a FOTA update package on the device through other methods.

This is a particularly nasty Attack if a device is vulnerable to it, in that it can completely change a device, perhaps without the user being aware that this is happening, and it can almost certainly be done remotely.

Suggested solutions:

- Make sure that only validated and signed data is used to re-flash the device
- Consideration must then be given as to where to store the validating and signature check code and data, and what the fallback position is if the current set of keys is leaked
- Make sure that only validated and signed code can be executed on the device as a result of re-flashing

Note: Current FOTA solutions claim this capability, and it may be considered that as they execute during a device re-boot, that their keys and software are validated and no opportunity to Attack them exists. It may be that only hardware Attacks can break current FOTA solutions, in which case why would a user allow their device to be so vulnerable, however many current Threats appear as Trojans and it may be in the future that such Trojan hardware modifications exist e.g. “This Mod Chip gives you free DRM (by breaking FOTA security)”.

T.SWM.004 Subverting of general software loading procedures	Moderate	Moderate	WWW

Description:

One place that the general OS may perform Integrity and Access Control checks is in the OS file loader. In this context, the file loader is the piece of code that allocates memory and then loads code from long term storage (Flash, hard drive etc) for execution as an executable, dynamically linked library or even driver. It may also inform the rest of the system as to the rights of that piece of code when it comes to dealing with APIs.

Therefore the file loader is a particularly critical piece of security code and if replaced with non-checking, full-access granting code, this would bypass much of any OS security.

By its very nature the file loader can often be considered a static piece of code (as nothing exists to move it around), hence its placement in memory can be carried out through secure boot.

Typically this means that it can only be Attacked by breaks in the user/privileged code split or through hardware techniques, allowing the file loader or information it is dependent upon to be replaced, or modified.

Suggested Solutions:

- Move this piece of code behind another layer of security boundary
- Perform checks to ensure that no code and static changes have been made to this piece of code
- If the file loader is outside the general security infrastructure (for performance reasons), perform remote load requests from the secure code base to see if the file loader security is running as expected

4.2 Software Opportunistic Threats (T.SWO.xxx)

Threat Description	Required Expertise	Ease of Repeat	Ease of Distribution
T.SWO.001 Attack via faulty OS code(bug)	Moderate	Easy	WWW

Description:

Recent studies have shown up to 33 security specific flaws per million lines of average code. (A long term study of the OpenBSD code base [2]).

Typical C code produces 17 bytes per line [3].

So for a typical 20Mb Feature phone there may be 40 security flaws.

This is not including bugs that just crash the system; these 40 are specifically security related flaws that leak secrets, break Access Control or the like.

Suggested Solutions:

- Reduce code size
- Intensify testing and reviewing on security critical code
- Isolate security critical code from general code base

T.SWO.002 Attack, via unrelated application APIs, to secure resources.	Moderate	Easy	WWW

Description:

Each and every API in an OS can potentially expose security vulnerabilities, even if it has nothing to do with security. This is because the OS APIs are the OS's separation mechanism between different domains of privileged data. That privileged data may be related to secrets, it may be related to methods of drawing on the screen, or it may be related to allocating memory and manipulating the MMU. Unfortunately these APIs are critical to exposing the rich and versatile functionality that general OS's need to provide. Adding security to all APIs slows down the OS and requires the re-writing of large areas of affected code.

Example of API vulnerabilities:

- OpenFile("C:\test.txt") is a legal API being used legally
- OpenFile("DBG:") is using that legitimate API, but making use of potential errors deeper down in the OS structure
- OpenFile("http://127.0.0.1/config.asp -SimUnlock") is using that legitimate API, but making use of potential errors deeper down in the OS structure
- OpenFile("%c%c") might cause an OS to report an error in the filename, but the error reporting code may break when displaying %c%c and display critical stack data. (%c is an internal command to the 'C' print instruction to display information that is on the stack)

Suggested Solutions:

- Use Type-Safe APIs so malformed data cannot be sent through them
- Isolate security critical code from the general code base, so that the boundary that must be crossed to enter that code is not the same one as that being crossed by general code API usage (e.g. the security

code is not in the same domain or sub-domain group as the graphics driver code)

T.SWO.003 Breaking Access Control performed by software to hardware features	Moderate	Easy	WWW

Description:

This is a sub-category of T.SWM.001 or T.SWO002 in which there is a flaw in an API. In this case the Attack is on an API that allows the Threat Agent to access hardware.

This often leads to further vulnerabilities as hardware has different access rights to software.

This vulnerability is increased in an environment where the APIs are forced to follow a schema created for a more general interface need. For example, general operating systems do not use Type-Safe structures to pass data across APIs because of the speed overhead, whereas a security specific API should ideally be using such structures

Suggested Solutions:

- Place critical hardware Access Control software in environment separated from general code base
- Consider if hardware (such as DMAs or CLCD controllers) can be used to circumvent other security (see T.SWO.008)
- Lock down some hardware capabilities so that they cannot be changed at the wrong 'time', e.g. system resources should only be allocated during boot
    - A good quality secure boot and system design should enable this
- Provide hardware and/or additional software features that restrict circumvented Access Control from exposing secrets
- Reduce size of security code to reduce incidence of bugs
- Perform detailed code analysis

T.SWO.004 Breaking Access Control performed by software to confidential data, code and keys	Moderate	Easy	WWW

Description:

This is a sub-category of T.SWM.001 or T.SWO002 in which a flaw in a security API allows the Threat Agent to access software or data. This vulnerability is increased in an environment where the APIs are forced to follow schema created for a more general interface need.

For example, general OS's do not use Type-Safe structures to pass data across APIs because of the speed overhead, whereas a security specific API should ideally be using such structures.

Suggested solutions:

- Deploy Type-Safe APIs across vulnerable boundaries to ease good practice
- Do not mix Type-Safe and non-Type-Safe APIs across the same API domain boundary
- Reduce the size of security code to reduce incidence of bugs
- Perform detailed code analysis

T.SWO.005 Buffer overflows	Moderate	Easy	WWW

Description:

This can be classified as the classic example of the vulnerability exposed through use of non-Type-Safe APIs (See T.SWO.004).

By passing data through an API where it is known that the API is designed to receive X bytes and we are passing X+N bytes, it can be arranged that the N bytes overflow into an area that was used by other storage (if it is a heap overflow) or will cause the return address to be corrupted (if it is a stack overflow).

Stack overflow allows the "Return" instructions (at the end of each block of code) to jump to random locations and therefore illegally run random code, or local data to be changed – for example modifying the local rights of a piece of code.

Heap overflow can allow injection of different data into other areas of the OS, and hence allow security control information to be modified.

Suggested solutions:

- Use a Type-Safe API; where all data has specified size information Buffer Overflow is prevented by buffer size tracking. Such tracking has a notable speed overhead.
- As above but only apply this to the boundary into security critical software
    - Note: it is of no value having a Type-Safe API across a domain boundary, if elsewhere there is a non-Type-Safe API across the same boundary, it will then be the non-Type-Safe boundary that

is being attacked, not the particular API functionality.			
T.SWO.006 Code verifiability related security holes	Moderate	Easy	WWW
Description: Bugs and security flaws exist in code, but it is possible to mitigate their occurrence. It is known that these bugs and flaws appear at certain rates based on design and testing of the code. Verification of code quality can be improved by the application of greater resources and techniques but is expensive. Suggested solutions: • Reduce the size of the code base • Isolate the security critical code into a small code base for concentrated analysis • Invite independent analysis • Deploy development regimes (coding standards and documentation methodologies) that lead to best practice • Provide execution environments that guard against flawed code by restricting the developer (reduced instruction set, controlled data import and export options)			
T.SWO.007 Unpredictable CPU instructions	Hard	Easy	WWW
Description: On a 32 bit RISC processor there are a potential 2^{32} instructions (in real systems it is actually more than this but a sensible figure is 4 billion instructions). Not all of these are defined. When an undefined instruction is executed the processor should perform an exception, but can potentially end up in an undefined state and therefore cause a security leak. It is also possible for the processor, through error or external interference, to find itself in a state that should not be reachable. For example: receiving security data into a register that has changed to a non-secure mode between the time the data request was made, and the time the actual data reaches the register. Suggested Solutions: • Do not use native code, where a Threat Agent may introduce such			

undefined instructions

- Use a processor where all native instruction options (including undefined) are verified in the RTL as being incapable of illegally breaking the secure execution boundary

T.SWO.008 DMA or CLCD use for accessing memories	Moderate	Easy	WWW

Description:

DMA (Direct Memory Access) is a method used to move data in the device independently of the CPU. As such it is not blocked by MMU level defences and the user/privileged split.

Such properties are typical also for other hardware modules that have “bus master” rights, so a similar Attack might be achieved via other “bus masters” such as another CPU, a CLCD controller, an ethernet controller, or even a camera block.

Suggested Solutions:

- Place access to the DMA system behind trustable software
- Restrict the memory the DMA system can access with hardware extensions (address space limiters, microprogrammable DMA, MMU platform extensions, etc.) so that DMA cannot be used to move unauthorised data and hence steal secrets

T.SWO.009 Faking of general software identity	Moderate	Easy	WWW

Description:

Typically general OS’s have some way of uniquely identifying an application. When software makes use of a security API, the protection offered may be broken by fooling that API as to the identity of the calling application. An example of this is an API that only will release keys to application ID 325, so application ID 666 breaks other (weaker) security in the environment to claim to be 325.

Suggested solutions:

- If the identity of an application is broken, then the general OS can be considered broken. The protection then is to only have exposed APIs that perform non-critical operations (i.e. don’t have an operation for retrieving a key once placed behind a security boundary). This means that services that make use of the key also have to be placed behind the same security boundary.

- To enable this form of defence and yet perform required tasks, it must be possible for an application to move actual code behind the security boundary, without endangering that boundary for other users.

T.SWO.010 CLCD use for displaying memories and interfering with displayed data	Moderate	Easy	WWW

Description:

This is a Threat similar to T.SWO.008. It is however separate because the information generated is immediately available outside the device, and so does not require so much in the way of supporting software.

The CLCD (Colour LCD) controller is the graphics chip in a mobile device. CLCD controllers are designed to be pointed at memory blocks which normally contain graphics data. This usually bypasses the CPU's MMU and any privileged / user protections it may offer.

For games, graphics chips typically have to be pointed at different memory banks by the game code. If no hardware protection exists then a 'game' can be used to display secrets on the screen. Interpreting the screen may be difficult as it is just 'raw' memory, but it is possible. It can be considered relatively easy Attack to perform, as graphics processors tend to allow this sort of capability through programmer-friendly APIs. However the translation of the data may be considered more difficult as a moderate knowledge of graphic display formatting is required.

Suggested Solutions:

- Use hardware means to block access from CLCD device to critical secrets
- Place Access Control software between the CLCD and applications to restrict target memory

T.SWO.011 Attack through uncontrolled API in general software space	Moderate	Easy	WWW

Description:

By uncontrolled, we are referring to APIs that are not designed from a security point of view. While they may be Type-Safe or Type-Unsafe, they should just be considered 'untrusted' if they have the potential to breach the security domain.

Suggested Solutions:

- Separate security and general API domains

T.SWO.012 Attack through uncontrolled instruction set space	Hard	Easy	WWW

Description:

If a program can issue commands to do anything, e.g. access any memory, then it is difficult to protect secrets from such a program. In a typical system, the user/privileged code split restricts the actual instructions available to user space native code, as well as the memory it can access. For example; user space native code cannot issue commands to switch off the user/privileged space separation.

This threat is realised by user space code breaking into the privileged space by some means, such as a Buffer Overflow attack.

Suggested Solutions:

- In any security code environment it is desirable to heavily control the functionality a programmer has to a limited set of instructions that are considered security safe

T.SWO.013 Attack through interaction of software concurrent processes causing logical breaks	Hard	Moderate	WWW

Description:

Malicious software could exploit scheduling concurrence to let a task access another task resources (e.g. where an operating system dynamically schedules processes that use common resources, by exploiting a race condition). This security problem is described in T.SWO.013.Suggested Solutions:

- Avoid any concurrence (i.e. with a single thread model)
- Stay with a simple thread interaction model in any area that is dealing with security, and isolate that area cleanly from other more complex processing models
- If using multi-threaded model, use an approach that allows the demonstration of robustness against the above threat.

T.SWO.014 Exploit software bugs in execution environment	Moderate	Easy	WWW
Description: Any software environment has the potential for bugs. Suggested Solutions: - Reduce the amount of code that can potentially access the environment containing the security critical code - Perform more validation and quality control effort on the security code compared to normal code environments - Get external validation of the code - Use developers who are aware of security and quality development issues. Special consideration should be given to selecting and training such developers, as in many instances prior experience may have conditioned them to produce rich, rather than secure, code			
T.SWO.015 Software Attack on Type Unsafe APIs inside the execution environment	Moderate	Easy	WWW
Description: The Threat is that even if the boundary between secure and non-secure environments is protected by Type-Safe APIs, that if there are Type-Unsafe APIs in the secure environment then these may cause security holes. Typically we are looking at limiting damage from bugs rather than hackers operating inside the security environment, though that should not be discounted completely. Suggested solution: Make use of Type-Safe APIs when possible inside a security environment. The in-built structural checking of the command data tends to reduce any effects of erroneous events			
T.SWO.016 Software Attack on Type-Safe APIs inside the Execution Environment	Hard	Easy	WWW

Description: The Threat is that even with Type-Safe APIs we may still see internal actions that are undesirable. In that case security should be guarded by consideration of what those Type-Safe APIs expose. For example, does the secure cryptography API have to work with keys, or can it work with handles to those keys, and leave the keys in secure storage. Suggested Solutions: • Design internal APIs to consider information hiding / access restriction wherever possible			
T.SWO.017 Attack Through Virtual Debug Port	Moderate	Easy	WWW
Description: A debug port need not be implemented as a hardware interface, but could be exposed as a software mechanism. A malicious application could then use that port for any attacks that could be attempted on a physical port, but with the ability for the attack to be distributed easily and executed remotely. Suggested Solutions: • Provide authentication mechanisms for access to the port • Add a hardware lock to prevent usage in the field • Ensure that the debug port cannot access security critical assets			

4.3 Hardware Threats – External (T.HWE.xxx)

Threat Description	Required Expertise	Ease of Repeat	Ease of Distribution
T.HWE.001 Unauthorised access via external invasive or non-invasive debug ports (e.g. JTAG, ETM)	Moderate	Moderate	Small Corp
Description: Security critical code and data may be held in memory. If that memory is accessible to the processor, then invasive or non-invasive debug ports can be used to extract that information. Invasive debug consists of a debug session that modifies or interrupts the program flow, such as the insertion of a breakpoint. Non-invasive debug does not alter program flow and is used to			

observe the operation of the processor, such as an embedded trace port.

Suggested Solutions:

- Provide hardware access restrictions to disable above debug ports when particular code is active and to block access to certain code space.
  - Such restriction could be enabled or disabled by hardware or software on the device, providing levels of invasive or non-invasive access depending on the rights of the party activating the debug.
- Do not provide invasive or non-invasive access to the device
  - This may not always be acceptable as it prevents pre and post deployment access required for realistic maintenance of devices.
- Ensure that the debug port cannot access security critical assets.

T.HWE.002 Unauthorised re-Flash of device through external debug port (e.g. JTAG)	Moderate	Moderate	Small Corp

Description:

This is one of the most common Attacks today. It is used to change IMEI and SIMlock (code and data), and also device type properties (making a cheap device activate the features of a more expensive phone). It occurs because manufacturers leave JTAG accessible on production devices to enable field upgrade and debug. The removal of connectors is typically bypassed by high street 'SIM Unlock' shops by placing in special jigs.

Suggested Solutions:

- Use security code or hardware to enable or disable JTAG for access to device
  - Typically based on flash rights certificates
- Ensure that re-flashed code and data cannot be used e.g. by checking authenticity and integrity during secure boot

T.HWE.003 Unauthorised re-flash of device through external serial interface	Moderate	Easy	Small Corp

Description:

This is the same as T.HWE.002, on devices which only have a serial port for debug and upgrade.

Suggested Solutions:

- Use security code or hardware to enable or disable serial port for access to the general code space
- Use security code or hardware to enable or disable serial port for access to the security code space
  - (Typically these two would receive debug rights certificates, and then turn on a specific set of capabilities based on these)
- Ensure that re-flashed code and data cannot be used e.g. by checking Authenticity and Integrity during secure boot

T.HWE.004 Bypass security by external battery removal	Easy	Easy	WWW

Description:

Should the external battery be the only source of power to a device, removing the supply during a sensitive operation may leave the UE in an unknown state. For example in a DRM situation, a song may have been played, but if the rights count is not decremented until the end of the song, then removing the power source could result in the user playing the song indefinitely.

In addition, by removing the power source for a sufficient amount of time, any internal clock may halt, and some long term memory may be corrupted.

Suggested Solutions:

- Ensure sensitive operations are performed in a sensible order
- Maintain some short term, on-device power source
- Check on board clocks for “sanity” at boot

T.HWE.005 Bypass security by external memory card removal	Easy	Easy	WWW

Description:

Similar to T.HWE.004, if an external memory card is used to hold rights to media, removing the card prior to rights manipulation may allow infinite plays.

Suggested Solutions:

- Ensure sensitive operations are performed in a sensible order
- Deploy a journaling file system

T.HWE.006 Scan chain attack (direct or side channel)	Moderate	Moderate	Small Corp

Description:

A scan chain is used to allow the contents of hardware registers in a SoC to be read and altered. If a secret is in current use by the device, the ability to read the scan chain exposes that secret. In addition, it may be possible to switch the core to a privileged state and then load secrets. However, such an attack is likely to require knowledge of the scan topology.

Suggested Solutions:

- Don't expose scan-chain directly
- Only enable scan-chain to authorized debugger
- Provide hardware access restrictions to disable scan chain when particular code is active
  - Such restriction could be enabled or disabled by hardware or software on the device, providing levels of invasive or non-invasive access depending on the rights of the party activating the debug
- Ensure that the scan chain does not encompass security critical assets

T.HWE.007 Built-in self-test	Moderate	Moderate	Small Corp

Description:

Built-in self test (BIST) allows a SoC to test its internal logic, by applying inputs from a known generator (usually a pseudorandom number generator such as a linear feedback shift register) and feeding the outputs through a multiple input signature register. The Threat arises from the ability to apply inputs to the system logic, which could be exploited to escalate privilege.

Suggested Solutions:

- Only enable scan-chain to authorized debugger
- Don't expose the BIST interface directly
  - Provide hardware access restrictions to disable scan-chain when particular code is active

4.4 Hardware Threats – Terminal Intrusive (T.HWT.xxx)

Threat Description	Required Expertise	Ease of Repeat	Ease of Distribution
T.HWT.001 Extract secret via Bus monitoring (hardware probes)	Moderate	Moderate	Small Corp

Description:

Any data that travels off-SoC to DRAM or flash is vulnerable to being stolen by applying a monitoring probe to the physical bus that it travels down. This could even be done to on-SoC busses (see T.HWC.001) but on-SoC monitoring is much harder as the probe must operate in the <60nm size area.

- Off-SoC monitoring may be performed without leaving a trace. Some security criteria do not expect the blocking of such Attacks, but do require that there should be some physical evidence left by such tampering
- On-SoC would probably destroy the device for normal use while preparing for the probing

Suggested Solutions:

- Route tracking sub-surface in the PCB to make access more difficult
- Keep the critical secret data and code on-SoC
- Apply protective layers to resist, or be indicative of, attempts to attach such probes
- Stack devices at package or die level to prevent attachment of probes
- Encrypt the data in transit over the bus

T.HWT.002 Unauthorised access via internal but off-SOC invasive or non-invasive Debug Ports (e.g. JTAG, ETM)	Moderate	Moderate	Small Corp

Description:

Security critical code and data may be held in memory. If that memory is accessible to the processor, then invasive and non-invasive debug ports can be used to extract that information.

Suggested Solutions:

- Provide hardware access restrictions to disable above debug ports when particular code is active and to block access to certain code space
  - Such restriction could be enabled or disabled by hardware or software on the device, providing levels of invasive and non-invasive debug access depending on the rights of the party activating the debug
- Do not provide invasive or non-invasive debug access to the device
  - This may not always be acceptable as it prevents pre and post deployment access required for realistic maintenance of devices

T.HWT.003 General hardware Attack on data in external RAM (e.g. Probing)	Hard	Moderate	Small Corp

Description:

Hardware probing of data held in external RAM. External RAM may often be probed via logic analyser, or be subject to data modification by a bus interception attack. This may expose any keys, data and code stored within, and data modification may be used to escalate privilege.

Suggested solutions:

- Do not store sensitive information off-SoC

T.HWT.004 Hardware Attacks on static information in internal RAM (on-SOC, outside package)	Hard	Hard	Small Corp

Description:

Hardware probing data held in internal RAM through exposed bus connections on the outside of the package which contains that RAM.

Typically information is discovered by having the device running and then exercising that information so it transits a bus that is being probed. This threat is typically used to extract class secrets, which may be stored on-chip.

See page 11 of "Advances in Smartcard Security", by Marc Witteman [4].

Suggested solutions:

- Do not allow external access to internal buses

T.HWT.005 Power analysis Attack to reveal secrets	Hard	Hard	WWW

Description:

By monitoring the variations in power consumption of a processor core it is possible to perform statistical analysis and deduce the actions of that core based on the power cost for various operations. Such Attacks have been performed in the past on smart cards and similar systems. While this works well on cores that do only one thing at a time, SoCs introduce noise to this operation from their many processor cores and delayed/phased bus activity. While this does not stop such an Attack, in theory it may make it much harder to actually perform.

Smartcards also get round this Attack by using onboard power regulation and power balancing systems.

These will tend to mean that a SoC will always run at the worst case power consumption, something not desirable in the general OS space. This will have a large battery impact on any device.

Suggested Solutions:

- Use smartcard technology to create your SoC
- Perform operations that must be protected against such Attacks in a separate or integrated smartcard type core

T.HWT.006 Time analysis Attack to reveal secrets	Hard	Hard	WWW

Description:

By monitoring the variations in power consumption of a processor core, and relating it to time, it is possible to perform statistical analysis and deduce the actions of that core based on the power cost for various operations. Such Attacks have been performed in the past on smart cards and similar systems. While this works well on cores that do only one thing at a time, SoCs' busses (with multiple delayed transactions) introduce noise to this operation due to having many processors and delayed/phased bus activity. While this does not stop such an Attack (in theory) it may make it much harder to actually perform.

Smartcards prevent this Attack by making all instructions take the same amount of time, and so no difference is available to be analysed. Obviously this means that all instructions must then take the length of time it would have taken for the worst case instruction. This will have a large performance impact on an application processor core, where the longest instructions (load and store multiple) can take 60 times the amount of time of the fastest.

Suggested Solutions:

- Use smartcard technology to create your SoC
- Perform operations that must be protected against such Attacks in a separate or integrated smartcard type core

T.HWT.007 Bypass security by glitch Attacks (e.g. power)	Moderate	Moderate	WWW

Description:

This is the technique of "glitching" the signals of the processor to cause it to perform actions other than those in its instruction sequence. The hope of the Threat Agent is that by doing so the processor will leak some data across a security boundary that will enable them to perform a wider spread security break. Typically a glitch will be induced by changing the supply voltage or clock frequency of a core, or by sending "out of specification" signals into random core pins.

Suggested Solutions:

- A full security core will have protection built into its structure to detect such Attacks and block leakage of information. Generally these protections are layout specific (i.e. design work done by the chip foundry or their physical design suppliers) and not related to the 'logic' of the processor
    - Such defences are not normally placed on a standard application processor, but there is no technical reason why some of these could not be considered

T.HWT.008 Bypass security by power removal to NV memory	Easy	Easy	WWW

Description:

Consider a system where long term security data is held by signing against secrets held in write-many non-volatile (NV) storage. What does that system do when NV storage fails?

It should be remembered that NV storage on-SoC is a difficult technology due to silicon manufacturing process considerations, and so often current NV memory is off-SoC and hence its data is vulnerable to interception.

Secure NV store often works by having a very small on-SoC NV counter plus an on-SoC Secret Key, linked algorithmically to a bigger off-SoC NV store such as Flash.

The on-SoC counter can be used to prove that the off-SoC store has not been tampered with. The question is then, what do we do when the on-SoC NV store fails?

Suggested Solutions:

- Off-SoC NV store works with a shared secret to protect the channel to the on-SoC secure execution space. Then no on-SoC NV storage may be needed

T.HWT.009 Attack through interaction of hardware concurrent processes causing logical breaks	Hard	Moderate	WWW

Description:

This threat is included because concurrency in a system makes it very difficult to perform any sort of security analysis. It is not a hole as such, but the parallel execution of unsynchronised actions makes it very hard to certify a system and therefore is not desirable. For example, where multiple CPU cores exist on a platform, software on one core may be able to affect execution of code on the other CPU by locking hardware resources. Alternatively, it is possible for two processors sharing a common cache to analyse the cache usage of the other processor to derive security information and even resolve keys.

Suggested Solutions:

- Stay with a simple thread interaction model in any area that is dealing with security, and isolate that area cleanly from other more complex processing models
- With a hardware Attack, one must make sure that the resolution of the information (for example in the cache usage) is insufficient to use statistical techniques to derive data
    - The concurrency in this case has to allow clearing of cache at frequencies on a par with the security operation cycle time and so is not practical Attack under a virtualisation model where transitions are in hundreds of cycles (or more)

4.5 Hardware Threats – Component Invasive (T.HWC.xxx)

In general there are a number of interesting papers focused on smartcards that are useful introductions to this topic, See Ref [4] and [5].

Threat Description	Required Expertise	Ease of Repeat	Ease of Distribution
T.HWC.001 Extract secret via Bus monitoring (de-cap/drill & hardware probes)	Hard	Hard	Small Corp

Description:

Applying a monitoring probe to a de-capped chip, or drilled multilayer PCB, allows access to physical busses that data traverses.

<table>
<tr><td colspan="4">• This is likely to destroy the device for normal use while preparing for the probing
Suggested Solutions:
• Obfuscate physical layout
• Apply protective layers to resist, or be indicative of, attempts to attach such probes.
• Stack devices at package or die level to prevent attachment of probes
• Encrypt the data in transit over the bus</td></tr>
<tr><td><b>T.HWC.002</b> Hardware Attacks on static information in internal RAM (on-SoC, inside IC package)</td><td>Hard</td><td>Hard</td><td>Small Corp</td></tr>
<tr><td colspan="4">Description:
Hardware probing data held in internal RAM. It is possible to use micromanipulator to place a probe onto a SoC and make contact with silicon buses that are in the surface layer of the silicon. A badly designed SoC might have a bus port allowing access to internal buses.
Static information has a discoverable location and so may be more susceptible to probing when multiple physical devices have to be destroyed to get one set of known good data as it provides a stable target for the Threat Agent. An example of this is eFuse, where a Threat Agent is looking to strip a SoC down to be able to physically see the state of an eFuse array. This threat is effective at extracting class secrets stored on-chip.
See page 11 of "Advances in Smartcard Security", by Marc Witteman [4].
Suggested solutions:
• Do not allow external access to internal buses
• Investigate technology that does not have a readily discernable physical presence.
o There are methods under development for creating device unique keys that do not require eFuse. Given a device unique key, then the device itself can encode, sign and store keys from external sources</td></tr>
<tr><td><b>T.HWC.003</b> Hardware Attacks on dynamic information in internal RAM (on-SoC, inside IC package)</td><td>Hard</td><td>Hard</td><td>Small Corp</td></tr>
</table>

<table>
<tr><td colspan="4">Description:

Hardware probing data held in internal RAM. It is possible to use a micromanipulator to place a probe onto a SoC and make contact with silicon buses that are in the surface layer of the silicon. A badly designed SoC might have a bus port allowing access to internal buses.

Dynamic information is more difficult to prove as authentic, as its signature (including both contents and location) may constantly change. However, this also makes an attack more difficult. This threat is effective at extracting class secrets stored on-chip.

Suggested solutions:

• Do not allow external access to internal buses
• Use a heuristic based integrity protection scheme to mitigate write attacks
• Use obfuscation techniques</td></tr>
<tr><td><b>T.HWC.004</b> De-capping of the chip holding secrets</td><td>Hard</td><td>Hard</td><td>Small Corp</td></tr>
<tr><td colspan="4">Description:

Using techniques such as etching with nitric acid (HNO_3), it is possible to remove the packaging from a SoC and allow Attacks on the silicon itself.

Suggested solutions:

• Place protective layers on the actual silicon that when removed break SoC operation.</td></tr>
<tr><td><b>T.HWC.005</b> Focused Ion Beam (FIB) manipulation</td><td>Hard</td><td>Hard</td><td>Small Corp</td></tr>
<tr><td colspan="4">Description:

A FIB is a variant of an electron microscope that can not only be used to examine chip surfaces but actually lay down deposits to change state and allow illegal attachment of probes.

This threat also includes scanning electron microscope attacks.

Suggested solutions:

• Apply the protective measures in T.HWC.004 and T.HWC.006</td></tr>
</table>

T.HWC.006 Probe Stations	Hard	Hard	Small Corp

Description:

Probe stations are available for Silicon manufacturers to diagnose production problems.

They can also be used to steal information that is local to a piece of silicon. Probing is normally used to read bus traffic and gather information that way.

Suggested solutions:

- Make on-SoC bus inaccessible by running sub surface
- Scramble the data travelling on-SoC

4.6 HARDWARE CLONING, COMPONENT REPLACEMENT OR COMPONENT ADDITION THREATS (T.CLO.XXX)

THREAT DESCRIPTION	REQUIRED EXPERTISE	EASE OF REPEAT	EASE OF DISTRIBUTION
T.CLO.001 Cloning Device by copying PCB and Flash	Hard	Hard	Corporate

Description:

Manufacturer "A" spends millions of dollars to develop a Device. The components are “off the shelf”, so the value is in the composition of those parts and the addition of the software they run. Manufacturer "B" can buy the same components, reverse engineer the PCB layout and make a copy of "A"s Flash to gain the software.

Suggested solution:

- This can be prevented if the SoC (a standard component) will not run the flash unless there is a common secret or key
    - If that common secret or key can be guarded from "B" then while the devices are identical "B" cannot make theirs run without changing the Flash
    - “B” cannot ‘ghost’ the RAM of a running device because if at any stage that ghost references information locked to that common key (which he doesn’t have) then it will fail
    - By ghosting, we are referring to taking a snapshot copy of the state of a device from RAM (using hardware emulation techniques on the external RAM), and starting the clone devices by dumping in that known state

T.CLO.002 External RAM Chip Replacement Attack	Hard	Moderate	Small Corp

Description:

Here we are considering replacement of an external RAM chip by another chip which may contain faked software or data, or have the capability to leak, manipulate or record data.

Suggested Solutions:

- Do not place secrets in external RAM
- Do not place secret manipulators in external RAM
- Encrypt and / or integrity protect the data in external RAM

T.CLO.003 Hardware Attacks to change static information in external RAM	Hard	Moderate	Small Corp

Description:

Once data is loaded from Flash to RAM for execution, it is typically considered only vulnerable to software Attacks such as software bug exploits. In the future, when protections are in place against that level of Attack, we will find that the sort of Mod Chip found on games consoles and DVD players will start to be seen on embedded devices. These typically consist of a microcontroller with limited ROM containing Attack code. They can take their power from the device, and are used to interfere with data on the bus between the SoC and the DRAM. As such they might also be used to Attack data travelling from Flash to SoC with Flashes that may be secure to normal programming /replacement Attacks.

Suggested Solutions:

- Route tracking sub-surface in the PCB to make access more difficult
- Apply protective layers to resist, or be indicative of, attempts to attach foreign hardware (mod chips)
- Stack devices at package or die level to prevent attachment of foreign hardware (mod chips)
- Move critical code and data into SoC package
- Implement verification of all code/data received from sources external to SoC
    - Consider the use of hardware based automatic verification

T.CLO.004 Hardware Attacks to change dynamic information in external RAM	Hard	Moderate	Small Corp

Description:

Here we are considering a Mod Chip that intercepts and changes varying code and data being written from SoC to random locations in RAM. This is a very advanced Attack and would only occur when simpler Attacks on the Flash to SoC and static code/data transfers have been blocked.

Suggested Solutions:

- While it is probably impossible to identify a change to dynamic data, it is possible to periodically test security critical interfaces to see if the guards are still functional, and hence to gather an indication of security failure
    - A typical example of this is to route the signal from the watchdog via the non-secure domain to the secure domain. This indicates that the schedulers in both domains are running, and that (importantly) messages are being correctly routed in the system
- Consider the use of hardware based automatic verification:
    - While it would be extremely difficult for software to keep track of ALL dynamic data being written to off-SoC memory, and verify it on return, it may be practical to build a hardware block that tracks these things on a page by page basis
    - If there was suitable hardware then to apply a 32bit CRC (Cyclic Redundancy Check - a fast simple hash) to 4Kbyte pages, it requires 1Kb of on-SoC memory to store the CRC's for each 1Mb of off-SoC memory. However such hardware is not simple and may add unacceptable system delay while it performs its CRC
- Route tracking sub-surface in the PCB to make access more difficult
- Apply protective layers to resist, or be indicative of, attempts to attach foreign hardware (mod chips)
- Stack devices at package or die level to prevent attachment of foreign hardware (mod chips)

T.CLO.005 Attack by replacement of Flash when power is off (pre-boot)	Hard	Moderate	Small Corp

Description:

If all routes for changing Flash contents are secured, through hacking 'defined' APIs such as JTAG / serial download / hacking software with its own Flash drivers, then it always possible to bypass all these by physically replacing the Flash with one containing hacked code.

Suggested Solutions:

- Secure boot may pick up this change as validation keys become incorrect
- Some Flashes contain unique IDs that can be linked to software that is built into the SoC
- Some Flashes now contain a processor that may provide internal protections against simple replacement

T.CLO.006 Attack by replacement of Flash when power is on (post-boot)	Hard	Moderate	Small Corp

Description:

Post-boot Flash substitution requires a little more effort by the Threat Agent.

An example of method that might be deployed is that of using a double size Flash device, with the same physical package outline as the original. It is then possible to boot from the normal address space with the usual static memory secure boot Integrity checks occurring. Then, at some time after completion of the boot process, the address space can be remapped to the upper half of the Flash simply by forcing the state of the highest address pin exposing the system to execute non-signed or unchecked code. Then the Attacking changes will take effect as soon as those code and data blocks are copied from Flash to RAM, for example by a paged memory system.

Suggested Solutions:

- Validate all data loaded from Flash against information that has its validation based in a source outside of Flash; do not just perform boot time validation
  - Note that on a typical 200Mhz processor it can take 0.2seconds to verify 2Mbyte of code - if you do not do anything else
- Employ the solutions suggested in T.CLO.005

5 DEFINITION OF TERMS

TERM	DESCRIPTION
ACCESS CONTROL	Security mechanism which guaranties Asset security properties by restricting the ability to use (read, execute, modify or delete) an Asset.
ASSET	Resource whose security properties need to be protected
ATTACK	An intentional act attempting to violate the UE Security Policy.
AUTHENTICITY	The property that a legitimate user can correctly identify an Asset as being genuine and attributable to its authors or caretakers
AUTHORISED PARTY	An entity that is authorised to perform a specific operation on Assets. This authorisation depends on the Security Policy of the Asset's owner (definition updated from 'OMTP Trusted Environment; OMTP TR0' [6])
BINDING	Binding is used to associate two or more entities to each other. The association makes it more difficult to without detection exchange one or more of the entities for entities which are not part of the original, valid association.
BUFFER OVERFLOW	A buffer overflow is an erroneous condition where a process attempts to store data beyond the boundaries of a fixed-length buffer. The result is that the extra data overwrites adjacent memory locations.
BUS	A collection of signals. In this document, often referring to the collection that will move data between a bus master and a memory (or peripheral) decoded to be at a specific address location
BUS MASTER	Any component (CPU, DSP, DMA, CLCD, etc) that is capable of initiating movement of data on a bus
CLASS SECRET	Any secret that is common to a group. These are particularly vulnerable due to availability on a large number of terminals. They are of a sensitive nature because the information gained can be used to attack a large number of devices.
EFUSE	Technology which allows for the dynamic real-time reprogramming of computer chips
EXTERNAL RAM	RAM which resides in a different IC package to the Bus Master that is communicating with it.
FLASH	process of re-programming a system

TERM	DESCRIPTION
INTEGRITY	The property of an Asset that it has been modified only by an Authorised Party (Definition updated from the ’OMTP Trusted Environment; OMTP TR0’ [6]).
INTERNAL RAM	RAM which resides in the same IC package as the Bus Master that is communicating with it.
MOD CHIPS	Equivalent to Modification Chips. small electronic device used to modify or disable built-in restrictions and limitations of a system
SCAN CHAIN	A method to enable a serial port on a SoC to connect directly to all registers and other locations holding dynamic state. Typically used in connection with debug, trace and loading of initial data into a new device.
SECURE BOOT	A method of starting a device that ensures that the code and data, that is initially used, comes from an authorised source.
THREAT	Capabilities, intentions and attack methods of adversaries, or any circumstance or event with the potential to violate or bypass the UE Security Policy.
THREAT AGENT	Any human user or Information Technology (IT) product or system, which may attempt to violate or bypass the UE Security Policy and perform an unauthorised operation with the UE.
TYPE-SAFE	Code that accesses only the memory locations it is authorized to access, and only in well-defined, allowable ways.
TYPE-UNSAFE	Code that is not Type-Safe

6 ABBREVIATIONS

ABBREVIATION	DESCRIPTION
API	Application Programming Interface
BIST	Built-in Self Test
CLCD	Colour Liquid Crystal Display
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
DRM	Digital Rights Management
DSP	Digital Signal Processor
ETM	Embedded Trace Macrocell
FIB	Focussed Ion Beam
FOTA	Firmware Over The Air
HTTP	Hyper Text Transport Protocol
HTTPS	Secure HTTP
IC	Integrated Circuit
IMEI	International Mobile Equipment Identity Number
I/O	Input / Output
JTAG	Joint Test Action Group
LCD	Liquid Crystal Display
MMU	Memory Management Unit
NV	Non-volatile

ABBREVIATION	DESCRIPTION
OS	Operating System
PCB	Printed Circuit Board
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RTL	Runtime Libraries
RTOS	Real-Time Operating System
SIM	Subscriber Identity Module
SoC	System-on-Chip
TCP/IP	Transmission Control Protocol / Internet Protocol
UE	User Equipment
VMM	Virtual Machine Monitor

7 REFERENCED DOCUMENTS

NO.	DOCUMENT	AUTHOR	DATE
1	Vocabulary for 3GPP Specifications (3GPP TR 21.905 v7.0.0) See http://pda.etsi.org/pda/home.asp?wkr=RTR/TSGS-0121905v640	3GPP	Sept 2005
2	Milk or wine: does software security improve with age?	Andy Ozment	2006
3	Estimating source lines of code from object code: Windows and Embedded Control Systems	Les Hatton	Aug 2005
4	Advances in Smartcard Security	Marc Witteman	July 2002
5	Design Principles for Tamper-Resistant Smartcard Processors	Oliver Kommerling Markus G. Kuhn	May 1999
6	Trusted Environment: OMTP TR0 http://www.omtp.org/Publications/Display.aspx?Id=f45e6775-2ecf-40fa-8cf2-dbe182ee9b58	OMTP	May 2009

------- END OF DOCUMENT --------