Sancus: Low-cost trustworthy extensible networked devices with a zero-software Trusted Computing Base

Job Noorman   Pieter Agten   Wilfried Daniels   Raoul Strackx
Anthony Van Herrewege   Christophe Huygens   Bart Preneel
Ingrid Verbauwhede   Frank Piessens

16 Aug 2013
Carna Botnet client distribution March to December 2012.

~420K Clients

Noorman et al.
Carna Botnet
Port scanning /0 using insecure embedded devices (Anonymous researcher)

Carna Botnet client distribution March to December 2012. ~420K Clients
Although very relevant, low-end devices lack effective security features

More threats on embedded devices
Due to network connectivity and third-party extensibility

No effective solutions exist
It’s “a mess” (Viega and Thompson)

Researchers are exploring this area
E.g., SMART (El Defrawy et al.)
Goal: design and implement a low-cost, extensible security architecture

Strong isolation of software modules
Given third-party extensibility

Secure communication and attestation
Both locally and remotely

Counteracting attackers with full control over infrastructural software
Zero-software Trusted Computing Base
Target: a generic system model

Infrastructure provider
*IP* owns and administers nodes *N*<sub>i</sub>

Software providers
*SP*<sub>j</sub> wants to use the infrastructure

Software modules
*SM*<sub>j,k</sub> is deployed by *SP*<sub>j</sub> on *N*<sub>i</sub>
Example node configuration

Node

\[ S \rightarrow SM_1 \rightarrow SP_1 \]

\[ SM_S \rightarrow SM_n \rightarrow SP_n \]

\[ \vdots \]

\[ \vdots \]

\[ IP \]
1. Module isolation
2. Key management
3. Remote attestation and secure communication
4. Secure linking
5. Results
Overview

1. Module isolation
   - Module layout
   - Access rights enforcement

2. Key management

3. Remote attestation and secure communication

4. Secure linking

5. Results
Modules are bipartite with a *public* text section and a *protected* data section

**Public text section**
Containing code and constants

**Protected data section**
Containing secret runtime data
Node with one software module loaded

- **Node**
  - Memory
    - Unprotected
      - Entry point
        - Code & constants
        - SM$_1$ text section
          - SM$_1$ protected data section
            - Protected data
              - Unprotected

- Protected storage area
  - $K_N$
  - $K_{N,SP,SM_1}$
  - SM$_1$ metadata
    - Layout
    - Keys

---

Noorman et al.  
Sancus  
16 Aug 2013  
10 / 29
Node with one software module loaded

Public and protected sections
Node with one software module loaded

Module layout

Node

Memory

Unprotected

Entry point

Code & constants

Unprotected

SM$_1$ text section

SM$_1$ protected data section

Protected data

Unprotected

Protected storage area

$K_N$

$SM_1$ metadata

$K_{N,SP,SM_1}$

Layout

Keys

Node Noorman et al. 16 Aug 2013 10 / 29
Node with one software module loaded

Module identity

Node

Memory

Unprotected

Entry point

$SM_1$ text section

Unprotected

$SM_1$ protected data section

Protected data

Unprotected

$SM_1$ metadata

Protected storage area

$K_N$
Node with one software module loaded

Module entry point

Node

Memory

Unprotected

Entry point

Code & constants

Unprotected

Protected data

Unprotected

Node

$SM_1$ text section

$SM_1$ protected data section

$SM_1$ metadata

Protected storage area

$K_N$

$K_{N,SP,SM_1}$

Layout

Keys

Noorman et al. Sancus

16 Aug 2013 10 / 29
Node with one software module loaded

Module keys

Node

<table>
<thead>
<tr>
<th>Memory</th>
<th>Unprotected</th>
<th>Entry point</th>
<th>Code &amp; constants</th>
<th>Unprotected</th>
<th>Protected data</th>
<th>Unprotected</th>
</tr>
</thead>
</table>

$SM_1$ text section

$SM_1$ protected data section

Protected storage area

$K_N, SP, SM_1$

$SM_1$ metadata

$K_N$
Modules are isolated using program-counter based memory access control

Variable access rights
Depending on the current program counter
Modules are isolated using *program-counter based memory access control*

Variable access rights
Depending on the current program counter

<table>
<thead>
<tr>
<th>From/to</th>
<th>Text</th>
<th>Protected</th>
<th>Unprotected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>r-x</td>
<td>r-x</td>
<td>rw-</td>
</tr>
<tr>
<td>Other</td>
<td>r-x</td>
<td>r--</td>
<td>---</td>
</tr>
</tbody>
</table>
Modules are isolated using *program-counter based memory access control*

Variable access rights
Depending on the current program counter

<table>
<thead>
<tr>
<th>From/to</th>
<th>Text</th>
<th>Protected</th>
<th>Unprotected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>r-x</td>
<td>r-x</td>
<td>rw-</td>
</tr>
<tr>
<td>Other</td>
<td></td>
<td>r-x</td>
<td>rwx</td>
</tr>
</tbody>
</table>

From/to | Text   | Protected | Unprotected |
---------|--------|-----------|-------------|
Text     | r-x    | r-x       | rw-         |
Other    |        | r-x       | rwx         |
Modules are isolated using *program-counter based memory access control*

**Variable access rights**

Depending on the current program counter

<table>
<thead>
<tr>
<th>From/to</th>
<th>Text</th>
<th>Protected</th>
<th>Unprotected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>r-x</td>
<td>r-x</td>
<td>rw-</td>
</tr>
<tr>
<td>Other</td>
<td>r--</td>
<td>---</td>
<td>rwx</td>
</tr>
</tbody>
</table>
Modules are isolated using
*program-counter based memory access control*

Variable access rights
Depending on the current program counter

Isolation of data
Only accessible from text section

<table>
<thead>
<tr>
<th>From/to</th>
<th>Text</th>
<th>Protected</th>
<th>Unprotected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td>rw-</td>
</tr>
<tr>
<td>Other</td>
<td></td>
<td></td>
<td>---</td>
</tr>
</tbody>
</table>
Modules are isolated using **program-counter based memory access control**

**Variable access rights**
Depending on the current program counter

**Isolation of data**
Only accessible from text section

**Protection against code misuse (e.g., ROP)**

<table>
<thead>
<tr>
<th>From/to</th>
<th>Text</th>
<th>Protected</th>
<th>Unprotected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>r-x</td>
<td>rw-</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>r--</td>
<td>---</td>
<td></td>
</tr>
</tbody>
</table>
Node with one software module loaded

Module entry point

Node

Memory

Unprotected

Entry point

Code & constants

Unprotected

Protected data

Unprotected

SM\textsubscript{1} text section

SM\textsubscript{1} protected data section

Protected storage area

K\textsubscript{N}

K\textsubscript{N,SP,SM\textsubscript{1}}

SM\textsubscript{1} metadata

Layout

Keys

Noorman et al.

Sancus

16 Aug 2013 11 / 29
Modules are isolated using program-counter based memory access control

Variable access rights
Depending on the current program counter

Isolation of data
Only accessible from text section

Protection against code misuse (e.g., ROP)
Enter module through single entry point

<table>
<thead>
<tr>
<th>From/to</th>
<th>Text</th>
<th>Protected</th>
<th>Unprotected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry</td>
<td>r-x</td>
<td>rw-</td>
<td></td>
</tr>
<tr>
<td>Text</td>
<td>r-x</td>
<td>rw-</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>r--</td>
<td>---</td>
<td></td>
</tr>
</tbody>
</table>
Modules are isolated using
*program-counter based memory access control*

Variable access rights
Depending on the current program counter

Isolation of data
Only accessible from text section

Protection against code misuse (e.g., ROP)
Enter module through single entry point

<table>
<thead>
<tr>
<th>From/to</th>
<th>Entry</th>
<th>Text</th>
<th>Protected</th>
<th>Unprotected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry</td>
<td>r-x</td>
<td>r-x</td>
<td>rw-</td>
<td></td>
</tr>
<tr>
<td>Text</td>
<td>r-x</td>
<td>r-x</td>
<td>rw-</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>r-x</td>
<td>r--</td>
<td>---</td>
<td></td>
</tr>
</tbody>
</table>
Modules are isolated using program-counter based memory access control

Variable access rights
Depending on the current program counter

Isolation of data
Only accessible from text section

Protection against code misuse (e.g., ROP)
Enter module through single entry point

<table>
<thead>
<tr>
<th>From/to</th>
<th>Entry</th>
<th>Text</th>
<th>Protected</th>
<th>Unprotected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry</td>
<td>r-x</td>
<td>r-x</td>
<td>rw-</td>
<td>rwx</td>
</tr>
<tr>
<td>Text</td>
<td>r-x</td>
<td>r-x</td>
<td>rw-</td>
<td>rwx</td>
</tr>
<tr>
<td>Other</td>
<td>r-x</td>
<td>r--</td>
<td>---</td>
<td>rwx</td>
</tr>
</tbody>
</table>
Isolation can be enabled/disabled using new instructions
Node with one software module loaded

Module layout

![Module layout diagram]

- **Node**: The central component of the module layout.
- **Memory**: The area where data is stored, divided into unprotected and protected sections.
  - **Unprotected**: Includes the entry point, code, and constants.
  - **Protected data**: Contains protected data sections.
- **Protected storage area**: A secured section within the memory, containing metadata.
  - **Metadata**: Includes layout and keys areas, protected with keys $K_N, SP, SM_1$.

The diagram illustrates the layout of the software module, emphasizing the separation of code, constants, and data into protected and unprotected regions.
Isolation can be enabled/disabled using new instructions

**protect** `layout, SP`
Enables isolation at `layout`

**unprotect**
Disables isolation of current SM
Overview

1. Module isolation

2. Key management

3. Remote attestation and secure communication

4. Secure linking

5. Results
Providing a flexible, inexpensive way for secure communication

Establish a shared secret
Between $SP$ and its module $SM$

Use symmetric crypto
Public-key is too expensive for low-cost nodes

Ability to deploy modules without $IP$ intervening
After initial registration, that is
Key derivation scheme allowing both Sancus and SP’s to get the same key

Infrastructure provider is trusted party
Able to derive all keys
Key derivation scheme allowing both Sancus and SP’s to get the same key

Infrastructure provider is trusted party
Able to derive all keys

Every node $N$ stores a key $K_N$
Generated at random
Key derivation scheme allowing both Sancus and SP’s to get the same key

Infrastructure provider is trusted party
Able to derive all keys

Every node $N$ stores a key $K_N$
Generated at random

Derived key based on SP ID
$K_{SP} = kdf(K_N, SP)$
Key derivation scheme allowing both Sancus and SP’s to get the same key

Infrastructure provider is trusted party
Able to derive all keys

Every node $N$ stores a key $K_N$
Generated at random

Derived key based on SP ID
$K_{SP} = kdf(K_N, SP)$

Derived key based on SM identity
$K_{SM} = kdf(K_{SP}, SM)$
Node with one software module loaded

Module identity

Node

Memory

Unprotected

Entry point

Code & constants

Unprotected

Protected data

Unprotected

$SM_1$ protected data section

$SM_1$ text section

Protected storage area

$K_N, SP, SM_1$

$SM_1$ metadata

Layout

Keys

Node Noorman et al. 16 Aug 2013 15 / 29
Node with one software module loaded

Module keys

![Diagram of memory layout with nodes, sections, and keys]

- **Node**: The primary unit of execution.
- **Memory**: Contains various sections and storage areas.
- **Unprotected** sections: Generally accessible to all modules.
- **Protected data section**: Contains sensitive information.
- **Unprotected** entry point: Entry point for code execution.
- **Code & constants**: Includes compiled instructions and constants.
- **Protected storage area**: Contains metadata for secure storage.
- **Layout** and **Keys**: Structures for organizing data and keys.
Isolation can be enabled/disabled using new instructions

\textbf{protect} \textit{layout, SP}

Enables isolation at \textit{layout} and calculates $K_{N,SP,SM}$

\textbf{unprotect}

Disables isolation of current SM
Overview

1. Module isolation

2. Key management

3. Remote attestation and secure communication
   - Key idea
   - Secure communication
   - Remote attestation

4. Secure linking

5. Results
Ability to use $K_{N,SP,SM}$ proves the integrity and isolation of $SM$ deployed by $SP$ on $N$

Only $N$ and $SP$ can calculate $K_{N,SP,SM}$
$N$ knows $K_N$ and $SP$ knows $K_{SP}$

$K_{N,SP,SM}$ is calculated after enabling isolation
No isolation, no key; no integrity, wrong key

Only $SM$ on $N$ is allowed to use $K_{N,SP,SM}$
Enforced through special instructions
Secure communication is provided by calculating MACs using the module key
Secure communication is provided by calculating MACs using the module key

\[ \text{mac}(K_N, \text{SP}, \text{SM}, \text{No}, I) \]
Secure communication is provided by calculating MACs using the module key

\[
N = \text{Calculate } O
\]

\[
\text{No, } I
\]
Secure communication is provided by calculating MACs using the module key

MAC is calculated by a `mac-seal` instruction
Using the key of the calling `SM`
Secure communication is provided by calculating MACs using the module key

MAC is calculated by a mac-seal instruction
Using the key of the calling SM

MAC can be recalculated by SP...
He knows the correct $K_N,SP,SM$
Ability to use $K_{N,SP,SM}$ proves the integrity and isolation of $SM$ deployed by $SP$ on $N$

Only $N$ and $SP$ can calculate $K_{N,SP,SM}$
$N$ knows $K_N$ and $SP$ knows $K_{SP}$

$K_{N,SP,SM}$ is calculated after enabling isolation
No isolation, no key; no integrity, wrong key

Only $SM$ on $N$ is allowed to use $K_{N,SP,SM}$
Enforced through special instructions
Secure communication is provided by calculating MACs using the module key

MAC is calculated by a `mac-seal` instruction
Using the key of the calling `SM`

MAC can be recalculated by `SP`...
He knows the correct $K_{N,SP,SM}$

...providing trust in the authenticity of messages
Only `SM` can create the correct MAC
Remote attestation is provided through secure communication

Attest integrity, isolation and liveliness

Of SM by SP
Remote attestation is provided through secure communication

Attest integrity, isolation and liveliness
Of SM by SP

Integrity and isolation attested by MAC, liveliness by nonce
Thus included in secure communication
Remote attestation is provided through secure communication

Attest integrity, isolation and liveliness
Of SM by SP

Integrity and isolation attested by MAC, liveliness by nonce
Thus included in secure communication

⇒ remote attestation ⊂ secure communication
So can be achieved more easily
Overview

1. Module isolation
2. Key management
3. Remote attestation and secure communication

4. Secure linking
   - Goals
   - Verifying modules
   - Optimizing multiple calls

5. Results
Enabling efficient and secure local inter-module function calls

Verify the $SM$ that is to be called
Is it the correct, isolated $SM$?

Inherently different from secure communication
May belong to different $SP$s; no shared secret

We can rely on protected local state
Gives rise to interesting optimizations
Modules are verified by calculating a MAC over their identity

Module $A$ wants to call module $B$

$A$ is deployed with a MAC of $B$’s identity using $A$’s key
In an unprotected section since it is unforgeable
Modules are verified by calculating a MAC over their identity

Module $A$ wants to call module $B$

$A$ is deployed with a MAC of $B$’s identity using $A$’s key
In an unprotected section since it is unforgeable

$A$ calculates the MAC of $B$’s actual identity
If they match $B$ can safely be called
Modules are verified by calculating a MAC over their identity

Module $A$ wants to call module $B$

$A$ is deployed with a MAC of $B$’s identity using $A$’s key.
In an unprotected section since it is unforgeable.

$A$ calculates the MAC of $B$’s actual identity.
If they match $B$ can safely be called.

Done through new instruction: `mac-verify`
Need assurance on $B$’s isolation.
The expensive MAC calculation is needed only once

We only need to know if the same module is still there
After initial verification, that is

mac-verify returns the ID of the verified module
Can be stored in the protected section
Later calls can use a new instruction:
get-id
Check if the same module is still loaded
The expensive MAC calculation is needed only once

We only need to know if the same module is still there
After initial verification, that is

Sancus assigns unique IDs to modules
Never reused within a boot-cycle

mac-verify returns the ID of the verified module
Can be stored in the protected section

Later calls can use a new instruction: get-id
Check if the same module is still loaded
Overview

1 Module isolation

2 Key management

3 Remote attestation and secure communication

4 Secure linking

5 Results
   - Hardware implementation
   - Module compilation
   - Evaluation
Complete implementation of Sancus based on the MSP430 architecture

Based on the openMSP430 project
Very mature open-source MSP430 implementation

Built on existing primitives:
- MAC: HMAC
- KDF: HKDF
- Hashing: SPONGENT-128/128/8 (Bogdanov et al.)

Usable in RTL simulator and FPGA
For easy testability of Sancus
Automatically handling the intricacies of compiling Sancus modules

Placing the runtime stack in the protected section
Prevent access by untrusted code

Clearing registers on module exit
Prevent data leakage

Supporting more than one entry point
Dispatching through a single entry point
Automatically handling the intricacies of compiling Sancus modules

```c
#include <sancus/sm_support.h>
#define ID "foo"

int SM_DATA(ID) protected_data;
void SM_FUNC(ID) internal_function() { /* ... */}
void SM_ENTRY(ID) entry_point() { /* ... */}
```
No runtime overhead on “normal” code; moderate overhead given enough computation

No impact on maximum frequency
Critical path not affected

Main overhead from calculating MACs
For verification and output

Smaller overhead from entry and exit code
Stack switching, register clearing,…
Example node configuration
No runtime overhead on “normal” code; moderate overhead given enough computation.
Area overhead

Fixed overhead: 586 registers / 1,138 LUTs
Mainly MAC and KDF

Per module: 213 registers / 307 LUTs
Mainly key storage
Review

1. Module isolation
   Isolation using *program-counter based access control*

2. Key management
   Hierarchical scheme with keys based on module’s *identity*

3. Remote attestation and secure communication
   Attestation based on the ability to use a key

4. Secure linking
   Module verification based on MAC of its identity

5. Results
   Simulator, FPGA and automatic compilation
Sancus: Low-cost trustworthy extensible networked devices with a zero-software Trusted Computing Base

Job Noorman  Pieter Agten  Wilfried Daniels  Raoul Strackx
Anthony Van Herrewege  Christophe Huygens  Bart Preneel
Ingrid Verbauwhede  Frank Piessens

https://distrinet.cs.kuleuven.be/software/sancus/