Digital Investigation 11 (2014) S66–S75

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

Digital investigations for IPv6-based Wireless Sensor Networks

Vijay Kumar [a], George Oikonomou [a, *], Theo Tryfonas [a], Dan Page [a], Iain Phillips [b]

[a] Crypto Group, University of Bristol, Faculty of Engineering, Merchant Venturers Building, Woodland Road, Bristol, BS8 1UB, UK
[b] Loughborough University, Computer Science, LE11 3TU, Loughborough, UK

Keywords:
Internet of Things
Wireless Sensor Networks
RAM and flash memory extraction
RAM content analysis
Contiki Operating System
Wireless sensor forensics

A B S T R A C T

Developments in the field of Wireless Sensor Networks (WSNs) and the Internet of Things (IoT) mean that sensor devices can now be uniquely identified using an IPv6 address and, if suitably connected, can be directly reached from the Internet. This has a series of advantages but also introduces new security vulnerabilities and exposes sensor deployments to attack. A compromised Internet host can send malicious information to the system and trigger incorrect actions. Should an attack take place, post-incident analysis can reveal information about the state of the network at the time of the attack and ultimately provide clues about the tools used to implement it, or about the attackr's identity. In this paper we critically assess and analyse information retrieved from a device used for IoT networking, in order to identify the factors which may have contributed to a security breach. To achieve this, we present an approach for the extraction of RAM and flash contents from a sensor node. Subsequently, we analyse extracted network connectivity information and we investigate the possibility of correlating information gathered from multiple devices in order to reconstruct the network topology. Further, we discuss experiments and analyse how much information can be retrieved in different scenarios. Our major contribution is a mechanism for the extraction, analysis and correlation of forensic data for IPv6-based WSN deployments, accompanied by a tool which can analyse RAM dumps from devices running the Contiki Operating System (OS) and powered by 8051-based, 8-bit micro-controllers.

Introduction

The 802.15.4 standard for low-power wireless communications, published by the Institute of Electrical and Electronics Engineers (IEEE), is among the key building blocks of Wireless Sensor Network (WSN) deployments. Recent research and standardisation efforts in the area have resulted in the adoption of protocols of the TCP/IP family for networks of severely constrained devices. For instance, IPv6 over Low Power Wireless Personal Area Networks (6LoWPAN) (Montenegro et al., 2007) and related Internet Engineering Task Force (IETF) specifications (Hui (Ed.) & Thubert, 2011; Shelby (Ed.) et al., 2012) have made it possible to use IPv6 in networks of embedded smart objects. For those networks, the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) (Winter (Ed.) et al., 2012) is the de-facto standard for routing.

These developments have resulted in IoT deployments where each node is uniquely identified by an IPv6 address and is directly attached to the Internet. As a result,

* Corresponding author. Tel.: +44 117 9545131.
E-mail addresses: vk12122@my.bristol.ac.uk (V. Kumar), G.Oikonomou@bristol.ac.uk (G. Oikonomou), Theo.Tryfonas@bristol.ac.uk (T. Tryfonas), Daniel.Page@bristol.ac.uk (D. Page), I.W.Phillips@lboro.ac.uk (I. Phillips).

http://dx.doi.org/10.1016/j.diin.2014.05.005
1742-2876/© 2014 Digital Forensics Research Workshop. Published by Elsevier Ltd. All rights reserved.

networks of smart embedded objects are exposed to new types of attack. In case of an incident, post-hoc analysis can potentially reveal the mechanism used to implement the attack and expose some details about the attacker's identity or location. Due to the constrained nature of IoT devices in terms of computational and storage capacity and due to network bandwidth limitations, logging is limited and traditional forensic techniques are not directly applicable.

The Contiki embedded Operating System for the IoT features a standards-compliant TCP/IP stack, with support for IPv6, 6LoWPAN, RPL and other relevant specifications. It is thus a suitable platform for this research.

In this context, this paper contributes the following:

1. We present a mechanism to extract the RAM contents of devices running Contiki.
2. We present a method for the analysis of extracted data and for the automated retrieval of network-related information, such as routing table and Neighbour Discovery (ND) cache contents.
3. We demonstrate the capability to correlate information gathered from multiple devices, in order to partially reconstruct the network topology.

Background and related work

Among existing research efforts in the field of forensics for WSNs is a remote live forensic protection framework, which prevents the execution of tampered software on a sensor node and informs other network nodes about an intrusion as soon as a device gets tampered with (Zaharis et al., 2010). The same work presents a sand-boxing technique to restrict memory access of a running application within a legitimate memory space, and techniques to prevent malicious code from execution by validating the authenticity of the running application.

Triki et al. (2009) propose a solution for digital investigation of wormhole attacks by creating a network of powerful observer or investigator nodes. Observers are responsible for the generation of information regarding sensor node behaviour and of forwarding this information to the network's base station. The authors also propose a set of algorithms to analyse evidence gathered at the base station, in order to identify colluding nodes and to reconstruct potential wormhole attack scenarios.

It has been demonstrated that a working prototype of a digital forensic readiness layer can be added over an existing IEEE 802.15.4 wireless sensor deployment (Mouton and Venter, 2011). Similar to the aforementioned work (Triki et al., 2009), the authors propose the addition of powerful forensic nodes in the network. Those nodes capture all data transmitted by normal network nodes and maintain data packet authenticity and integrity. This work mainly focuses on the reduction of time and cost incurred by digital investigations and on the ability to collect evidence without modifications to an existing network.

A solution based on more powerful observer nodes has also been investigated (Rekhis and Boudriga, 2009). Thanks to increased processing capacity, these observers are capable of analysing the patterns of various types of attack, such as wormhole, black-hole, sink-hole and sybil. To reduce overhead, captured traffic is processed on observers, and only illegitimate behaviour is sent to the base station.

In a slightly different approach, it has been demonstrated that compressed sensing techniques can be employed in order to overhear encrypted wireless transmissions, detect the traffic's periodic components and ultimately reveal the type of application deployed in the network (Fragkiadakis and Askoxylakis, 2013). The authors discuss the attacker's side, but similar principles could be adopted to conduct traffic analysis for forensic purposes.

Most research efforts discussed above gather evidence by sniffing wireless traffic on an overlay network of observer nodes. Setting up the observer network incurs additional cost and imposes increased network overhead. Furthermore, the work presented by Triki et al. (2009) detects only worm-hole attacks, the approach proposed by Rekhis and Boudriga (2009) detects worm-hole, black-hole and sybil attacks, while the research contributed by Mouton and Venter (2011) provided a framework for integrity and authenticity of packets. The discussion in Zaharis et al. (2010) mainly targets the detection of physical attacks, with RAM dumps used only to verify the address space, but without yielding any additional information. Our work aims to address that gap by searching RAM dumps for artefacts which can be used as evidence.

Since our work does not rely on traffic analysis, it does not require an overlay network of observer nodes, thus saving the additional installation cost. Communication overhead is not added either. The work presented here is complementary to efforts based on traffic analysis.

Relevant IETF specifications

6LoWPAN is defined by the IETF in Request for Comments (RFC) 4919 (Kushalnagar et al., 2007), 4944 (Montenegro et al., 2007) and 6282 (Hui (Ed.) & Thubert, 2011). The main objective of these specifications is to optimise transmission of Internet Protocol Version 6 (IPv6) datagrams over IEEE 802.15.4 radio links. The need for such optimisations arises due to the maximum layer 2 frame size in IEEE 802.15.4 networks, which is only 127 octets including the Media Access Control (MAC) header. A typical User Datagram Protocol (UDP) datagram over IPv6 requires 8 bytes for the UDP header and 40 bytes for the fixed IPv6 header. Including the layer 2 header and potentially additional IPv6 extension headers results in an overhead of approximately 65 bytes or more in each packet. In other words, 50% or more of each frame is consumed by header overhead. To address this situation, 6LoWPAN defines header compression techniques through an adaptation layer, alongside a mechanism for datagram fragmentation and reassembly.

For 6LoWPANs, the de-facto standard routing protocol is RPL. It is a distance vector protocol, which perceives the 6LoWPAN as a tree-like structure called a Destination Oriented Directed Acyclic Graph (DODAG). The DODAG is created based on a combination of metrics and constraints known as objective functions and which are used to calculate the best path between a source and destination. The graph building process is initiated at an administratively configured node, which is essentially the tree's root and is often referred to as a *Border Router.*

Data traffic in an RPL network can flow upwards in the tree (from a node towards the root), while support for downward flow of data traffic is optional. For point to point communication (from any node to any node), datagrams first travel upwards until they reach a node which is common ancestor to both the source and destination. They are then forwarded downwards to their destination.

RPL specifies a set of Internet Control Message Protocol Version 6 (ICMPv6) control messages. i) DODAG Information Solicitation (DIS), ii) DODAG Information Object (DIO), iii) DODAG Destination Advertisement Object (DAO) and iv) DAO Acknowledgement (DAO-ACK). Nodes use them to exchange graph-related information.

ACPO guidelines

The Association of Chief Police Officers (ACPO) has published a set of guidelines to deal with investigations related with computer-based crime (Association of Chief Police Officers, 2011). They involve four stages of evidence recovery: collection, examination, analysis and reporting. Additionally, no data held on a computer or storage media should be changed by any action of a law-enforcement agency. When access to the original data is required, the persons handling the data should be competent to do so and should record an explanation of the relevance and implications of their actions. An audit trail of the forensic process should be documented so that when a third party repeats the process they should be able to reproduce the same result. The person responsible for the investigation is responsible for the handling of the evidence and ensuring that these laws are adhered to. These recommendations and principles should be followed while doing any computer-related crime investigation.

Design and implementation

Our work can be broken down into the following steps:

1. *Extraction*: The retrieval of a copy of device RAM and flash contents.
2. *Analysis*: The examination of retrieved data.
3. *Co-relation*: If multiple devices are being investigated, additional information may be retrieved by co-relating data retrieved from different devices.

Hardware and software

To implement and test our work we have used sensor devices powered by the Contiki[1] open source embedded OS for the IoT. To conduct our tests, we deployed a 6LoWPAN network in our labs, we allowed it to operate under normal traffic conditions and subsequently we performed the steps discussed previously. Our testbed is formed by a combination of the following hardware platforms:

- *RPL nodes*: A number of Texas Instruments (TI) SmartRF05 Evaluation Boards with CC2530 Evaluation Modules and Sensinode N740 NanoSensors.
- *RPL Root*: A TI CC2531 USB dongle.
- *Wireless Sniffer*: A Sensinode N601 NanoRouter for live traffic capture and analysis, primarily used for network debugging.

The Texas Instruments devices are powered by the CC2530 System-on-Chip (SoC), with a Micro Controller Unit (MCU), 256 KB of flash, 8 KB of volatile RAM, a radio transceiver and various peripherals packaged in a single chip (Texas Instruments, 2012). The Sensinode devices are powered by the older but very similar CC2430 SoC (128 KB flash, 8 KB RAM) (Texas Instruments, 2007). The MCUs on all aforementioned devices are Intel 8051 derivatives. All our testbed devices are supported by a Contiki OS branch, which specifically targets 8051-based architectures (Oikonomou and Phillips, 2011).

In terms of software, our work uses the following tools:

- *Toolchain*: The Small Device C Compiler,[2] an ANSI compliant C toolchain for embedded devices, including devices powered by Intel 8051 CPUs, such as those used in our testbed. It is used to build Contiki as well as all embedded components of our work.
- *Sensinode/CC2430 Flash and RAM manipulation*: BooTTY is a minimalistic bootloader which allows users to manipulate the flash of Sensinode devices over a Universal Asynchronous Receiver/Transmitter (UART) interface and Ball is its host-side counterpart. We originally developed them for node Over-Air Programming, but their capabilities combined with BooTTY's very small code footprint made them very suitable for this work with small modifications.

Alternatively, it is possible to extract flash contents from Sensinode devices using a tool called “Nano USB Programmer”. This tool is used to program nodes and was provided by the manufacturer at the time of hardware purchase.

- *CC2530 Flash and RAM manipulation*: We used CC-Tool,[3] an open source software which connects to 8051-based SoCs over a debug interface. It can be used to manipulate node flash, for instance to program new firmware or read its contents. We modified this tool to add RAM dumping functionality.

Extraction of flash and RAM contents

Flash and RAM contents can be extracted from devices using the aforementioned tools. For CC2530 devices,

[1] http://www.contiki-os.org.

[2] http://sdcc.sourceforge.net/.

[3] http://sourceforge.net/projects/cctool/.

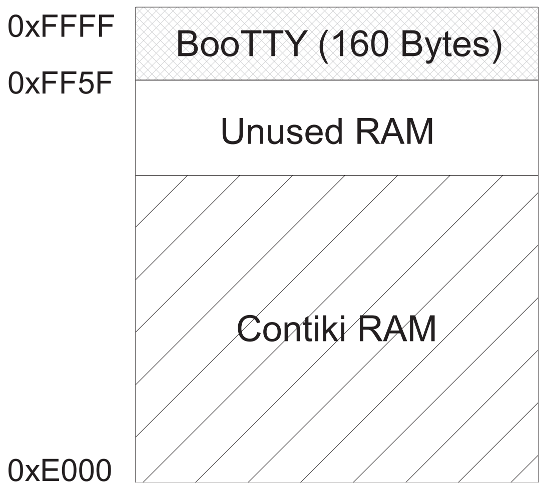


Fig. 1. CC2430 memory map on a node programmed with Contiki and BooTTY.

extraction is performed with CC-Tool. For Sensinode devices, two options are available: i) With Nano USB Programmer or ii) With BooTTY/Ball. The former has the advantage that it will work without modifications to the device. The downside is that it cannot be used for RAM extraction. Conversely, BooTTY provides RAM extraction capabilities but requires modifications to the device after seizure, since the investigator will have to program the node with the BooTTY bootloader. Conceptually, this is not dissimilar to rooting an Android phone during forensic investigation, which is common practice (Andriotis et al., 2012).

This approach has the following implications. Firstly, programming the node with a bootloader will partially overwrite previous flash contents. The CC2430 flash memory is organised in 64 pages of 2048 bytes each. Overwriting a part of the CC2430 built-in flash is only possible by first erasing the area to be re-written, with pages being the smallest erasable unit (Texas Instruments, 2007). As discussed previously, BooTTY has a very small code footprint: it only requires a single page on the device's flash. In other words, to use the bootloader the modification to node flash is the smallest possible.

Secondly, the bootloader itself requires approximately 160 bytes of RAM out of a total of 8 KB available on the SoC. As discussed above, nodes under investigation are operating with the Contiki OS, which has been built with the SDCC toolchain. Unless the developer dictates otherwise, variable allocation on RAM with SDCC starts from low addresses on the memory map (0xE000 for the CC2430) and increments towards address 0xFFFF. Therefore, a Contiki image with a 7 KB RAM footprint will occupy the lower 7 KB of the device's RAM. Keeping that in mind, we have allocated BooTTY's variables in high RAM addresses. Therefore, unless a Contiki image has a very large memory footprint (larger than 8192 − 160 bytes) the bootloader will operate without overwriting Contiki data, as illustrated in Fig. 1.

In terms of open issues, all of the above approaches require access to the hardware's debug interface. More specifically, CC-Tool and Nano USB Programmer communicate with the devices directly over the debug interface. With BooTTY, communication takes place over the UART interface. However, the investigator would first need to program the device with the bootloader, which also requires access to the debug interface. This restriction has some implications which are further discussed in Section Open issues.

ACPO compliant retrieval

For RAM and flash content extraction we followed the four stages of forensic course of action: seizure (device retrieval), examination (gathering information about the make, model, data-sheets, debugger tools), analysis (searching for network information, RAM carving), reporting (creation of a forensic report for each device).

Especially in the case of extraction from Sensinode devices using BooTTY/Ball, we consider two scenarios:

1. *Co-operative Device Owner.* In this use case, the owner of the infrastructure has agreed to pre-load devices with the bootloader to facilitate an investigation. In this case, extraction can take place without changes to the evidence.
2. *Non co-operative Device Owner.* In this use case, the investigator first has to program the device with BooTTY. Thus, principle two of computer-based electronic evidence should be adhered to: The investigator must give proper explanations and document the relevance and implications of the bootloader installation process. Changes caused by the investigator's actions will not be considered as evidence.

Consequently, the pre-installation of a bootloader is considered an important step towards the achievement of forensic-readiness of a 6LoWPAN deployment.

In case of SmartRF05 EB 2530 EM, since we are extracting RAM directly over the debug interface, we are not changing the evidence. We adhere to all the four principles applicable on dealing with computer based evidence.

Analysis

Analysis of flash contents

Contiki's source code is maintained on GitHub[4] and source version control is conducted with the git[5] distributed version control system. Every time a node boots, during startup Contiki prints a string like this:

```
Contiki-2.6-562-g299a292
```

This string follows this pattern:

```
Contiki-<tag>-<num-commits>-g<hash>
```

[4] https://github.com/contiki-os/contiki.

[5] http://git-scm.com/.

This pattern can reveal some information about the version of the source code used to build the image:

1. The most recent git tag that can be reached from the version of the source code is `<tag>`.
2. The version of the source code is `<num-commits>` newer than `<tag>`.
3. The exact version of the source code used to build the image, as represented in the git history, was `<hash>`. Note the presence of a literal ‘g’ character before the hash, which is not part of it.

This string is stored verbatim on flash and its retrieval is trivial; it is sufficient to open the dump with a hex editor or perform a search with a unix shell one-liner.

If the firmware developer obtained Contiki's sources through git and built the image without applying any changes to the source tree, the investigator can obtain a copy of the exact sources used. With access to the sources, the investigator can examine the characteristics of data structures, such as fields, offsets, data sizes, data types. Additionally, if the investigators know the version of the tool-chain used, they can build an identical image and retrieve from the assembled code and symbol table the exact RAM location of data structures. With that information available, they can directly inspect specific memory locations of interest, instead of having to rely on zero-knowledge RAM carving techniques.

Analysis of RAM dumps

As part of this work, we developed a tool to automate the analysis of RAM dumps taken from nodes equipped with an 8051-based SoC. Even though some patterns are visible by human eye and can be manually extracted, an automated method can significantly speed up the process and therefore allow investigators to apply their domain expertise to less tedious tasks.

Our tool fully automates the discovery of networking information present in RAM by following the steps discussed in the following paragraphs. Those steps are performed sequentially. Step 1 requires a full search of the entire RAM contents, step 2 requires a further two full passes and lastly one more full pass takes place during step 3. All remaining steps are based on previously identified RAM locations and do not require any further full searches. Since RAM size is only 8 KBs, each pass takes very little time, even less so for the remaining steps. This is further discussed in Section Evaluation.

During the discussion in the following paragraphs, we use the notion of a ‘*valid pointer*’. All nodes used in this work are equipped with an Intel 8051-based MCU and therefore have multiple, discrete, but partially overlapping memory spaces. For the CC2430 and CC2530 SoCs, those memory spaces are DATA, XDATA, SFR and CODE (Texas Instruments, 2007, 2012). The exact mapping of each memory space and variable allocation strategy depend on multiple factors, such as hardware design, compiler extensions and command line arguments and a full description is out of context of this paper. The entire discussion in this section refers to variables allocated to the XDATA memory space, which maps the entire physical RAM among other things. With this in mind, consider the following pointer declaration:

```
int *foo;
```

In SDCC terminology, this is a ‘*generic*’ pointer and occupies 3 bytes in RAM. The most significant byte holds the memory space the pointer refers to and the remaining two bytes store an address within that memory space. In this context, a 3-byte sequence is a ‘*valid pointer*’ if:

- The MSB contains the value `0x00`, which signifies that this is a pointer to XDATA and
- The address stored in the two remaining bytes is within the sub-region of the XDATA space which corresponds to RAM. On the CC2430 SoCs, RAM is mapped in the XDATA range [`0xE000`, `0xFFFF`]. On the CC2530, RAM is mapped between `0x0000` and `0x1FFF` inclusive.
- NULL pointers are treated as ‘*valid*’ in this context.

1. Link-Local Unicast Addresses: We start the process with a search for link-local addresses (`FE80::/64`), which are stored in RAM as 16-byte data chunks starting with `FE 80 00 00 00 00 00 00`. We store all candidate link-local addresses as well as their location in RAM, which is required in subsequent steps. These link-local addresses may belong to the node under investigation, but they may also belong to one of its network neighbours.

2. Global Unicast Addresses: Each of the link-local IPv6 addresses identified during step 1 has a 64-bit wide host suffix, which is automatically generated based on a network interface's Extended Unique Identifier identifier (EUI-64). The same suffix is used for the interface's global IPv6 addresses. At this step, we search the RAM for all 16 byte occurrences ending with this 8-byte suffix and *not* starting with `FE 80 00 00 00 00 00 00`. All matches are potential global IPv6 addresses which may or may not belong to the node under investigation. The first 8 bytes of those matches are candidate network prefixes. Having identified candidate network prefixes, we search the entire RAM for one last time for 16-byte blocks starting with one of those candidate network prefixes. Any such block can also be a global IPv6 address. This step is likely to introduce false negatives; this is discussed in Section Evaluation.

3. EUI-64 Identifiers: As discussed above, link-local and global IPv6 addresses are generated based on interface EUI-64 identifiers. During this auto-generation, the 7th most significant bit of the EUI-64 gets inverted. During the previous steps, we have identified all candidate IPv6 interface suffixes. At this step, we search for 8-byte patterns which match the interface suffix, but which have the 7th most significant bit inverted. These are candidate EUI-64 identifiers.

4. Neighbour Cache (ND Cache): In Contiki, each entry in the ND cache is a data structure containing the neighbour's link-local IPv6 address, its EUI-64 identifier and a series of meta data, such as the entry's lifetime and status. The cache itself is a statically allocated array of entries. An excerpt of the C struct used to declare entries is displayed in Listing 1, which also displays the definition of a cache which holds a maximum of `SIZE` neighbours.

```
typedef struct uip_ds6_nbr {
  uint8_t isused;
  uip_ipaddr_t ipaddr;
  uip_lladdr_t lladdr;
  /* Meta Data */
} uip_ds6_nbr_t;

uip_ds6_nbr_t uip_ds6_nbr_cache[SIZE];
```

Listing 1. An entry in Contiki's ND Cache.

The important observation is that the neighbour's IPv6 address (`ipaddr`) and EUI-64 identifier (`lladdr`) are stored in adjacent RAM locations; the corresponding starting addresses will have a 16 byte offset. From previous steps, we have stored the RAM offsets containing link-local IPv6 addresses and EUI-64 identifiers. It is therefore trivial to locate the ND cache by comparing the offsets between those locations. Furthermore, since each entry has a fixed size (let this be called LEN), this adjacency pattern will be repeated every LEN bytes up to a maximum of SIZE − 1 times. Therefore, we can identify the boundaries of the ND Cache, as displayed in Fig. 2.

5. Node's Own EUI-64 Identifier: Any EUI-64 identified during step 3, but which is not encountered inside the ND cache (step 4) belongs to the node under investigation.

6. Node's Own IPv6 Addresses: Among link-local and global IPv6 addresses already identified during steps 1 and 2, those belonging to the node under investigation will have a host suffix formed by the node's own EUI-64 identifier (step 5) with the 7th most significant bit inverted.

7. Default Route: In a RPL network, routes towards the root of the tree (‘upward’ routes) are stored as a default route, which is represented by the data structure displayed in Listing 2. Observe that entries form part of a linked list.

```
typedef struct uip_ds6_defrt {
  struct uip_ds6_defrt *next;
  uip_ipaddr_t ipaddr;
  struct stimer lifetime;
  uint8_t isinfinite;
} uip_ds6_defrt_t;
```

Listing 2. An entry in Contiki's Default Routes Table.

To find default routes, we use previously identified link-local address locations (step 1). A link-local address is part of a default route if i) it is not one the node's own addresses (step 6) and ii) the 3 bytes immediately preceding it in RAM constitute a *valid pointer* (`next` field). The RPL instance table also contains pointers to default routes, which is used for verification during subsequent stages.

8. RPL Parents: A parent in the DODAG is a node's immediate successor in the path towards the root (Winter (Ed.) et al., 2012). An excerpt of the declaration of the data structure representing a RPL parent is displayed in Listing 3. Each entry starts with two generic (3-byte) pointers (`next` and `dag`), followed by the metric container (`mc`) data structure (7 bytes), followed by a link-local IPv6 address. This address may not be part of the ND Cache (step 4) and may not be among the node's own addresses (step 6). To locate RPL parents, we search RAM regions around link-local addresses for memory blocks starting with two contiguous valid pointers, followed by any 7 bytes, followed by the link-local address.

```
struct rpl_parent {
  struct rpl_parent *next;
  struct rpl_dag *dag;
  rpl_metric_container_t mc;
  uip_ipaddr_t addr;
  rpl_rank_t rank;
  /* more fields */
};
```

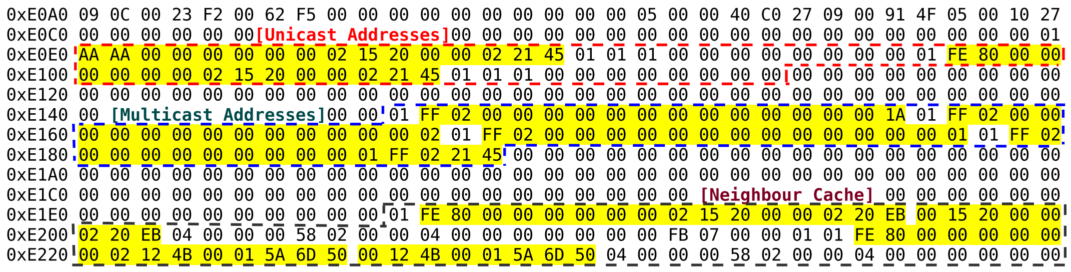
Listing 3. Extract of the RPL parent data structure.

9. RPL Instance Table and DODAGs: Information about RPL instances and DODAGs is nested. Instances are stored in a statically allocated array, with each element containing information about an individual instance. Within each instance, there is a sub-array holding information about the different DODAGs associated with it. Additionally, each DODAG data structure contains a pointer which can be used to obtain a reverse reference to the instance hosting it.

Locating RPL DODAG information is straightforward, since each RPL parent (step 8) holds a pointer to the DODAG it belongs to (field dag in Listing 3). DODAG data structures can only be encountered in RAM as part of an entry in the instance table. As soon as a DODAG is located, the relevant element in the instance table can be found by following the reverse reference discussed above.

10. Network Prefixes: A network prefix is an 8-byte sequence which can appear on RAM either by itself or as the 8 high bytes of a global IPv6 address. The prefixes of the networks a node belongs to are stored as part of DODAG data structures, which have been identified during step 9. Therefore, retrieval of network prefixes is trivial.

11. Routing Table: Routes to a specific destination are stored in a separate data structure from the one used for



```
0xE0A0 09 0C 00 23 F2 00 62 F5 00 00 00 00 00 00 00 00 00 00 05 00 00 40 C0 27 09 00 91 4F 05 00 10 27
0xE0C0 00 00 00 00 00 00[Unicast Addresses]00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01
0xE0E0 AA AA 00 00 00 00 00 00 02 15 20 00 00 02 21 45 01 01 01 00 00 00 00 00 00 00 00 01 FE 80 00 00
0xE100 00 00 00 00 02 15 20 00 00 02 21 45 01 01 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0xE120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0xE140 00 [Multicast Addresses]00 00 01 FF 02 00 00 00 00 00 00 00 00 00 00 00 00 00 1A 01 FF 02 00 00
0xE160 00 00 00 00 00 00 00 00 00 00 00 02 01 FF 02 00 00 00 00 00 00 00 00 00 00 00 00 00 01 01 FF 02
0xE180 00 00 00 00 00 00 00 00 00 01 FF 02 21 45 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0xE1A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0xE1C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [Neighbour Cache] 00 00 00 00 00 00
0xE1E0 00 00 00 00 00 00 00 00 00 00 01 FE 80 00 00 00 00 00 00 02 15 20 00 00 02 20 EB 00 15 20 00 00
0xE200 02 20 EB 04 00 00 00 58 02 00 00 04 00 00 00 00 00 00 00 FB 07 00 00 01 01 FE 80 00 00 00 00 00
0xE220 00 02 12 4B 00 01 5A 6D 50 00 12 4B 00 01 5A 6D 50 04 00 00 00 58 02 00 00 04 00 00 00 00 00 00
```

Fig. 2. Extract of a RAM dump. The ND Cache and Interface Tables are enclosed in dashed lines. Highlights illustrate IPv6 addresses and EUI-64 identifiers.

default routes. The relevant part of the data structure is displayed in Listing 4. Each row in the routing table contains a (destination, nexthop) tuple, which is represented by two IPv6 addresses stored in the structure's fields `ipaddr` and `nexthop` respectively. The former is a global IPv6 address, while the latter is link-local. Neither can be among the node's own addresses. During steps 1 and 2 we have identified the location of all IPv6 addresses in RAM and we can now search among them to identify pairs stored in adjacent locations. The 3 bytes preceding each such pair must constitute a valid pointer.

```
typedef struct uip_ds6_route {
  struct uip_ds6_route *next;
  uip_ipaddr_t ipaddr;
  uip_ipaddr_t nexthop;
  /* more fields */
} uip_ds6_route_t;
```

Listing 4. An entry in Contiki's Routing Table.

12. Network Interface: It holds the network interface's operational parameters and information about the IPv6 addresses associated with it. Addresses are stored in three separate arrays, one per type of address (unicast, anycast, multicast). Each array element holds a single address preceded by a single-byte flag to signify whether the element is currently in use (`isused`). Additional metadata are stored for each unicast address, resulting in a total RAM occupancy of 28 bytes per address.

To identify the interface data structure, we search near locations holding the node's own IPv6 addresses (step 6). Because of RAM footprint characteristics discussed above, two IPv6 addresses with an offset of 28 bytes between each other are likely to be part of the interface structure. Additionally: i) the byte directly preceding each of those addresses is the isused byte and must have the value 1, ii) the 3 following bytes are `state`, `type` and isinfinite. `state` can only have values 0, 1 or 2. `type` can only have values 0, 1, 2 or 3. Lastly, `isinfinite` can only be 0 or 1. Once the boundaries of the unicast addresses array have been identified, we search for multicast addresses relying on the fact that each address starts with `0xFF` and is preceded by the `isused` byte, which must equal 1.

Correlation among multiple RAM dumps

Extracting information from a single node can aid an investigator determine the node's networking state. It is of interest to see whether correlating RAM dumps extracted from multiple nodes can provide additional information and whether the network topology can be reconstructed through this correlation. Information found in ND caches can help us reconstruct a layer 2 network topology (Fig. 3), while information from the routing table and default routes can assist with the generation of a network graph at layer 3 (Fig. 4). In both figures, each ellipsis corresponds to a node and edges illustrate connectivity. Numbers represent the 4 high bytes of the node's IPv6 addresses. Observe how some arrows seem to be missing in Fig. 3, for instance the arrow from `:6d50` to `:2145`. This is due to the fact that we generated the graph based on partial information, since we only extracted RAM dumps from a subset of nodes in our lab deployment.

Due to limitations posed by the number of nodes at our disposal, the findings discussed in this section are only preliminary. However, they do show that this kind of corelation is feasible, that our approach is fundamentally correct and that it works for the link layer as well as for layer 3.

Experimental results and evaluation

For experimentation and evaluation we set up a small testbed in a lab environment, using the tools discussed in Section Hardware and software. In a fashion similar to what has been discussed in Section ACPO compliant retrieval, we consider two use-cases, depending on whether the owner of the deployment is co-operative with the investigator or not. In the former case, the investigator will have at their disposal additional information, such as the exact version of sources used to build device images and the resulting memory map and symbol tables. Additionally, nodes will be pre-programmed with a bootloader (if applicable) and the debug interface will be unlocked.

RAM retention and impact of device restarts

The CC2430 and CC2530 SoCs are equipped with 8 KB of Static RAM (SRAM). To achieve low energy consumption, devices of both types can operate in four different Power Modes (PMs) numbered from 0 to 3. The CC2530's SRAM

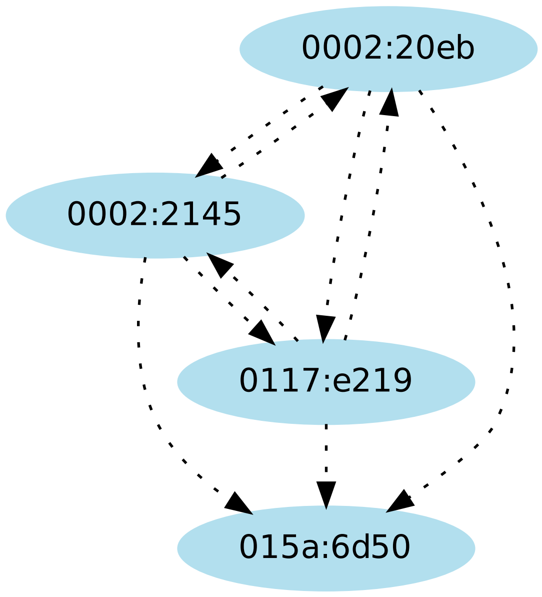


Fig. 3. Layer 2 network information.

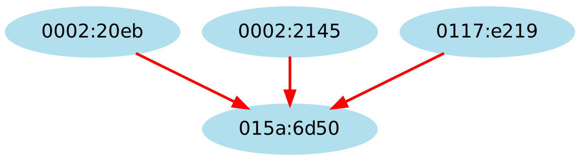


Fig. 4. Layer 3 network information.

contents are retained under all power modes, but approximately half SRAM contents of the CC2430 are lost under PMs 2 and 3 (Texas Instruments, 2007, 2012). According to the same documentation, SRAM contents are undefined after power-on for both SoCs.

In order to test this, we programmed Sensinode devices with a firmware which allows us to trigger a watchdog reboot by pressing one of the general purpose buttons on the node. After starting the devices and letting the network operate for a few minutes, we then pressed the button triggering a soft reset and afterwards we immediately retrieved RAM contents. In this scenario, we found that the extracted image was intact and contained salvageable data. Sensinode devices are also equipped with an on/off switch, which we used with a similar process to cause a full power-cycle instead of a soft reset. In this case, after a power-cycle the RAM's non-retention area contained garbage, and we could only recover useful data from the areas that retain contents in all power modes.

During its boot process, Contiki populates data structures with initial values (e.g. all zeros), overwriting evidence. As discussed above, a device reset will not necessarily destroy data, but it is important to prevent Contki from booting. If the device is programmed with BooTTY, Contiki startup will be prevented. In all other scenarios, it is important to extract data immediately after seizure and without resetting the device, in order to prevent data structure initialisation. In an operational deployment, devices will normally be battery-powered and the above requirement should be possible to meet. To achieve this, an investigator will have to connect the device to a PC and immediately start the extraction without power-cycling it.

Table 1 summarises the data extraction process' level of success under different scenarios. *Failure* signifies that Contiki restarted fully and set all variables to initial values, overwriting previous data. *Partial* indicates that RAM contents were partially lost due to RAM retention characteristics after a device reset.

Evaluation

As discussed in Section Analysis, the analyser tool parses the entire RAM four times. At the end of the last pass, all RAM locations of interest have been identified and stored. The remainder of the analysis is conducted by inspecting those areas only and is extremely fast. For example, the time to analyse 500 RAM dumps was approximately 2 min and 40 s on a typical desktop computer.

Our RAM carving mechanism can occasionally yield false positives, by flagging a memory block as an area of interest when in reality it is not. During our experimentation, this only occurred for small data structures, such as IPv6 addresses and EUI-64 identifiers. From what we observed in our experiments, the tool usually identifies between 10 and 15 candidate IPv6 addresses present in a RAM dump, including node's own addresses, as well as addresses present in the routing table and the neighbour cache. Among those, no more than 1 or 2 were false positives. Similar metrics apply in the case of EUI-64 identifiers. Those must subsequently be identified by applying the investigator's domain expertise, but given the small number of occurrences, we consider this to be a trivial task. In terms of the more complex data structures (e.g. the ND cache and RPL instance table) they adhere to very detailed rules and follow very specific patterns, making them unlikely to occur randomly. This is especially true for data structures which link to each other through pointers, for instance by being part of a linked list. If a number of memory locations have been identified as candidate matches, we verify that this is not a false negative by following next pointers and confirming that all matches are indeed part of a linked list. For data structures that are part of a pre-allocated array and occupy contiguous memory blocks, verification is based on offsets between consecutive occurrences.

Imaginary use-case

The GINSENG research project demonstrated that it is possible to use a WSN of Contiki-powered nodes to control mission-critical applications in an operational oil refinery (O'Donovan et al., 2013). Even though this effort used a bespoke network stack, the control systems industry has been considering a move to open standards, such as TCP/IP and web technologies (Byres and Lowe, 2004). Thus, future adoption of 6LoWPAN for industrial automation is not inconceivable.

In order to demonstrate our work's potential usefulness, we discuss an imaginary scenario of an industrial automation system, which uses wireless sensors to monitor and control environmental parameters and machinery. The sensors form a 6LoWPAN/RPL network and communicate, over IPv6, with a central server for data reporting, command and control. An attacker uses a similar sensor node to join the network by exploiting IPv6 ND or RPL vulnerabilities and subsequently induces malicious commands to the automation system, causing disruption, loss of availability and financial loss. The owner of the deployment has kept a record of the EUI-64 identifiers and IPv6 addresses of the nodes in the network. The network topology cannot be known a-priori, since RPL uses wireless link quality metrics to determine the best path between nodes and the network's root. Nevertheless, by using the techniques described in this paper, investigators can conclude that these commands did not originate from a node within the deployment, but from an EUI-64/IPv6 address combination assigned to an external, unauthorised device. Investigation of nearby premises then reveals a device with the EUI-64 identifier used to implement the attack. At an even later stage, by reverse engineering the firmware running on this device, it may even be possible to determine the exact method used to implement the attack.

Open issues

The CC2430 and CC2530 SoCs are equipped with a proprietary debugging interface (Texas Instruments, 2007, 2012); this allows operations such as programming the SoC flash, or single-stepped instruction execution. As discussed in Section Extraction of flash and RAM contents, extraction of RAM and flash contents requires access to this debug interface.

Table 1
Data Extraction Success Matrix. *Success* indicates that all information discussed in Section Analysis of RAM dumps was successfully recovered.

Device and use-case	Device not reset	Device soft-reset	Device power-cycled
Sensinode N740			
Bootloader Pre-Installed	Success	Success	Partial
Bootloader Installed by Investigator	Partial	Partial	Failure
SmartRF05EB + CC2530EM	Success	Data Overwritten	Data Overwritten

Unauthorised access or update is prevented by various sets of lock bits, also stored on the flash (within the flash information page). When the lock bits are enabled, the extraction process discussed in Section Extraction of flash and RAM contents will be impossible, since the only way to re-enable debugging is by erasing the contents of the entire flash (via so-called bulk erasure). Bypassing this mechanism is an issue we did not pursue, but that is clearly important: for co-operative device owners (who are required to pre-install BooTTY) it *may* be reasonable to leave the flash unlocked, but certainly not so for non co-operative owners.

Joye and Tunstall (2012, Chapter 16) survey fault injection techniques wrt. cryptographic targets, placing a general emphasis on faults during computation rather than in stored data. At least two strategies seem viable in theory:

1. With control over both the debug clock and chip power supply, glitch- and depletion-base (Joye & Tun-stall, 2012, Section 16.2.2) approaches could disrupt the bulk erase process (e.g., using a partial- and full-power supply for the main and information pages respectively, in an attempt to "skip" writes to the former and hence retain the content).
2. Targeted injection of an optical (Skorobogatov and Anderson, 2002; Skorobogatov, 2010) (e.g., laser) or EM-based stimulus could attempt to "bump" lock bits into a disabled state; this would typically imply chip depackaging (with the result classed as semi-invasive, though not *necessarily* destructive).

Realising such strategies concretely is challenging however. Even if successful, further questions relate to alignment with the ACPO guidelines: fault injection techniques are typically probabilistic, so it is possible evidence on the flash may be altered during the process in ways that are not easy to quantify or reproduce (even after profiling a replica device to maximise efficacy).

Support for different hardware and OSs

The current version of this work targets the Contiki Operating System. In order to conduct successful forensic analysis, it is common practice for tools to be aware of the Operating System under investigation. For example, Volatility[6] requires knowledge of the underlying OS in order to successfully analyse RAM dumps. Similarly, forensic tools for smartphones rely on knowing whether the OS under analysis is Android (and what version) or iOS. Currently, there are four dominant operating systems for embedded devices: Contiki, TinyOS, OpenWSN and FreeRTOS. To our knowledge, our hardware is only supported by Contiki. Therefore, in order to extend this work to different OSs, it would first have to be extended to support more hardware platforms. Within Contiki, the networking artifacts under investigation in this work are implemented in a platform-independent fashion and data structure declarations are identical. Even so, due to differences related to toolchains and MCU architectures, the following issues would have to be taken under consideration:

Data structure padding and alignment: This work investigates builds for 8-bit MCUs, and the toolchain will not pad data structures in this instance. To extend this work for 16- or 32-bit architectures, parts of the RAM carving algorithm would have to be re-written in order to take into consideration the possibility of padding and issues related to memory alignment. This is trivial.

Valid pointers: The notion of valid pointers discussed earlier in this paper only applies to SDCC builds for 8051-based MCUs. For architectures using a single memory space for instruction and data memory and where images are built using a gcc-derivative, the validity of pointers would relate to the start and end address of memory regions (.data, .bss, .text). For instance, a function pointer will normally be expected to point to within the boundaries of .text, whereas a pointer to a data structure in RAM will normally point to a location within the boundaries of the .data section. Nevertheless, even under a different definition, the notion of valid pointers will still be relevant and we believe that the differences discussed here would not be an insurmountable obstacle.

Extraction: For RAM content extraction, we rely on BooTTY and Ball for CC2430s and on a modified version of cc-tool for CC2530s. Embedded platforms with high market penetration are typically accompanied by similar open-source tools that can be used to program them. One such example is a python script distributed with Contiki and used to program the Cortex M3-based CC2538 SoCs. Those tools are good candidates for modification in order to achieve the goal of RAM extraction.

The task of extending this work to support additional operating systems would be more complicated. This is due to the fact that the implementations of 6LoWPAN, RPL and related specifications have considerable differences across different OSs. For example, TinyOS ships with the Berkeley Low-power IP stack (BLIP), which is an implementation of a number of IP-based protocols. Basic data structures such as IPv6 addresses and EUI-64 identifiers are likely to be identical, but we anticipate major differences in terms of routing tables, RPL-related data structures and the format of the ND cache.

Conclusion and future work

In this paper we have presented a method for the extraction and analysis of networking evidence from devices used in 6LoWPANs. As part of this work, we have

[6] https://code.google.com/p/volatility/.

developed a series of accompanying tools, including one which automates and facilitates analysis, thus allowing the investigator to focus on tasks which require domain expertise. The tool is reliable, fast and produces very few false positives. We also demonstrated that it is possible to partially reconstruct the network's layer 2 and 3 topologies, by correlating information extracted by only a subset of the nodes forming the network.

We now aim to investigate the requirements which need to be satisfied before we can reconstruct the full topology. In a large deployment of hundreds of devices, it will then be possible to achieve good results by investigating only a subset of carefully selected nodes, possibly in conjunction with traffic analysis techniques. Multiple image correlation with traffic analysis can also be used to detect specific attack patterns, such as Man in the Middle.

Investigations can be further facilitated by a logging infrastructure. It is possible to log some information on devices themselves, but this has drawbacks: Limited storage capacity means the logs can not be very detailed. Furthermore, logs will be distributed across the entire deployment and correlation will need to take place. It is also possible to implement a centralised logging infrastructure, for instance by using a syslog server. In this scenario, the drawback is that logging messages would increase network traffic and the approach would not scale well with deployment size and level of required logging detail. We aim to investigate hybrid approaches, whereby logs are cached locally and sent to a remote location periodically, possibly in an aggregated fashion.

For future work, we aim to extend our tool to support analysis of RAM dumps taken from devices of different architectures. We aim to support MSP430-based and ARM Cortex-based devices. The former due to their existing success and high market penetration and the latter because they are getting adopted by an increasing number of node manufacturers. Once these objectives have been achieved, our method will apply to a mix of 8, 16 and 32-bit MCUs of Harvard as well as Von Neumann architectures, offering very high technology coverage.

Acknowledgement

This work has been supported by the European Union's Prevention of and Fight against Crime Programme “Illegal Use of Internet” – ISEC 2010 Action Grants, grant ref. HOME/2010/ISEC/AG/INT-002.

References

Andriotis P, Oikonomou G, Tryfonas T. Forensic analysis of wireless networking evidence of android smartphones. In: Proc. IEEE International Workshop on Information Forensics and Security (WIFS 12); 2012. pp. 109–14. Tenerife, Spain.

Association of Chief Police Officers. Good practice guide for computer-based electronic evidence. V5.0; 2011.

Byres E, Lowe J. The myths and facts behind cyber security risks for industrial control systems. In: Proc. of the VDE Kongress, vol. 116; 2004.

Fragkiadakis A, Askoxylakis I. Malicious traffic analysis in wireless sensor networks using advanced signal processing techniques. In: Proc. 14th International Symposium and Workshops on a World of Wireless, Mobile and Multimedia Networks (WoWMoM); 2013. pp. 1–6.

Hui (Ed.) J, Thubert P. Compression format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. RFC 6282; 2011.

Joye M, Tunstall M, editors. Fault analysis in cryptography. Information security and cryptography. Springer; 2012.

Kushalnagar N, Montenegro G, Schumacher C. IETF RFC4919: IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): overview, assumptions, problem statement, and goals; 2007.

Montenegro G, Kushalnagar N, Hui JW, Culler DE. Transmission of IPv6 packets over IEEE 802.15.4 networks. RFC 4944; 2007.

Mouton F, Venter H. A prototype for achieving digital forensic readiness on wireless sensor networks. In: AFRICON, 2011; 2011. pp. 1–6.

O'Donovan T, Brown J, Büsching F, Cardoso A, Cecílio J, Ó JD, et al. The ginseng system for wireless monitoring and control: design and deployment experiences. ACM Trans Sen Netw 2013;10(4):1–4. 40.

Oikonomou G, Phillips I. Experiences from porting the contiki operating system to a popular hardware platform. In: Proc. 2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS); 2011. Barcelona, Spain.

Rekhis S, Boudriga N. Pattern-based digital investigation of x-hole attacks in wireless adhoc and sensor networks. In: Ultra Modern Telecommunications Workshops, 2009. ICUMT '09. International Conference on; 2009. pp. 1–8.

Shelby (Ed.) Z, Chakrabarti S, Nordmark E, Bormann C. Neighbor discovery optimization for low-power and lossy networks. RFC 6775; 2012.

Skorobogatov S. Flash memory “bumping” attacks. In: Cryptographic Hardware and Embedded Systems (CHES); 2010. pp. 158–72. Springer-Verlag LNCS 6225.

Skorobogatov S, Anderson R. Optical fault induction attacks. In: Cryptographic Hardware and Embedded Systems (CHES); 2002. pp. 2–12. Springer-Verlag LNCS 2523.

Texas Instruments. CC2430: A True System-on-Chip solution for 2.4 GHz IEEE 802.15.4/ZigBee®; 2007.

Texas Instruments. CC253x System-on-Chip Solution for 2.4-GHz IEEE 802.15.4 and ZigBee®Applications; 2012.

Triki B, Rekhis S, Boudriga N. Digital investigation of wormhole attacks in wireless sensor networks. In: Network Computing and Applications, 2009. NCA 2009. Eighth IEEE International Symposium on; 2009. pp. 179–86.

Winter (Ed.) T, Thubert (Ed.) P, Brandt A, Hui J, Kelsey R, Levis P, et al. IETF RFC6550: RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks; 2012.

Zaharis A, Martini AI, Perlepes L, Stamoulis G, Kikiras P. Live forensics framework for wireless sensor nodes using sandboxing. In: Proceedings of the 6th ACM workshop on QoS and security for wireless and mobile networks Q2SWinet '10. New York, NY, USA: ACM; 2010. pp. 70–7.