Forensic DHCP Information Extraction from Home Routers

Memory Forensics on SOHO / Enduser Embedded Routing and Gateway Systems

University of Amsterdam

Tobias Fiebig (tobias.fiebig@os3.nl)

August 6, 2013

Abstract

This document explores the feasibility and admissibility of a so far unrecognized source for digital evidence. The extraction of a suspect's home router's main memory to obtain valuable evidence is proposed and implemented. This method aims at providing time lines of devices appearing in a home network, and therefore possibly in that home, along with their owner. The technique is implemented and tested on the standards of modern volatile memory forensics. The results show that the proposed method is viable and may be extended to a wide range of devices.

Keywords: Volatile Memory; Forensics; DHCP; Home Router;

1 Introduction

At least in the European Union, the consumer broadband Internet access has seen a huge growth in recent years. Undoubtedly this growth will continue within reasonable bounds [25].

To enable an end-user to use multiple devices behind his designated broadband connection, most Western Internet Service Providers (ISPs) equip their customers with dedicated devices that handle the broadband connection as well as the distribution of the connection to the local network [1].

This is usually accomplished by employing a local RFC1918 [21] addressing schema, which is managed by the provided device using the Dynamic Host Configuration Protocol (DHCP) [6]. Connectivity to the Internet is then established by performing Network Address Translation (NAT) [24] on the provided device.

Common protocols for establishing the connections with local clients include IEEE 802.1(z) [19, 9] and IEEE 802.11(abgn) [32, 33, 35].

These devices are usually referenced as SOHO or End-user Routers, or, more fitting to their actual functionality of providing gateway services between the protocols used in the local network and the protocols used on the broadband link, as gateways. This document will further reference them just as *home routers*.

As home routers handle and control the local network traffic in their associated network segment, they may be a viable target during a forensic investigation. This document will explore these possibilities.

1.1 Related Work

Home routers usually employ either a MIPS [18] or ARM [10] based architecture [8] along with a dedicatedly tailored firmware. These devices are generally considered embedded devices.

Due to common challenges for embedded device development and necessities associated therewith, the Joint Test Action Group (JTAG) IEEE 1149.1 [16] was developed. Breeuwsma et. al. [4] investigated in 2006 how this port can be utilized for the forensic imaging of embedded devices and whether these ports are sufficiently accessible in available devices to allow for a common use of these during forensic investigations. They concluded that this method of access is a viable option for the acquisition of non volatile internal memory like for example the flash storage of a device [4].

In 2010, Roeloffs and Van Eijk [31] investigated the use of JTAG techniques to forensically extract the Random Access Memory (RAM) of TomTom GPS navigation systems [31]. Although they were unable to acquire large amounts of qualitative data, this is the first attempt of extracting RAM

[1] Called Customer Premise Equipment (CPE) by ISPs

from an embedded device via JTAG for forensic purposes known to the author of this document.

This spans the arc to memory forensics on home routers. While investigations on the extraction of forensic information from penetrated and abused home routers already have been conducted [28, 29] and a considerable amount of work has been put into the acquisition of network forensic information from home routers fully controlled by the investigator [15], a transfer of the techniques presented by Roeloffs and Van Eijk on TomTom systems to a general approach viable for home routers has not yet been conducted.

This research will attempt a first step into the direction of a universal forensic technique, allowing the utilization of volatile memory on home routers during a forensic investigation aimed at the owner of said device. To sufficiently scope the research at hand, the area of research will be limited to the investigation of the presence of specific devices within a Local Area Network (LAN) controlled by a single home router.

2 Hypothesis

Due to the known limitations of flash based storage [3] the author of this paper assumes that all relevant state information in regard to the previously mentioned RFC1918 [21] address propagation via DHCP [6] are kept in memory.

These state recordings, commonly called lease files, are necessary for the functionality of the DHCP. For each device associated with the network, the home router is responsible for they have to contain at least the Media Access Control (MAC) address of a client device as well as the time stamp of the last address assignment operation and the assigned Internet Protocol (IP) address [6].

Hence it is the author's hypothesis that from the information extracted from the main memory of a home router a timeline can be established, showing the association and disassociation of devices in a forensically viable manner.

Therefore this research will conduct a set of explorations and experiments allowing the verification of said hypothesis.

2.1 Contribution to the Field

If the proposed hypothesis can be validated, presence of a certain device at a certain place at a certain time can be established. In the light of recent developments in the mobile phone sector the presence of a specific mobile device in a network can be associated with the presence of the owner in the vicinity of the network devices.

Investigators may be able to utilize this information to indicate association between two persons, one being the owner of a certain identified device, the other one being the owner of the router device.

As this technique utilizes the JTAG port of a device, no cooperation from the legitimate owner of a home router is necessary.

If the approach succeeds, the door to further investigations of the memory contents of home router devices is opened.

3 Forensic Requirements

The restriction of conducting the proposed research in a forensically sound manner imposes various constraints on the methods that may be applied. While some are only relevant during a real criminal investigation and are composed mostly of procedures that have to be implemented during the handling of the evidence produced by and the data the evidence is produced of, some have to be incorporated into the method itself.

While these aspects will be briefly discussed here, the following section will cover how each of the constraints contemplated here has been addressed in the design process of the developed methodology.

If the requirements can not be implemented in the proposed method, this section will cover why this is not the case, and how a procedure that would have to be implemented during a real criminal investigation could satisfy the constraints nevertheless.

This summary of requirements is mostly based on the work of Vömel and Freiling [34], who recently published a detailed analysis of the constraints that have to be implemented during a volatile memory acquisition. Furthermore, an aspect of legal proceedings, the reproducibility of the extraction of evidence, will be covered.

3.1 Correctness and Completeness

The first constraint set for a forensically viable volatile memory image determined by Vömel and Freiling consists of correctness and completeness.

According to Vömel and Freiling, an acquired memory image may be considered complete and correct, if and only if all values set in the physical memory are conserved *as is*, that means unmodified, to the created image [34, p. 131], and if and only if it holds that all values that were present in the source physical memory if and only if they have been present in said source memory [34, p. 131].

Although an image may be partially correct but not complete, if it does not contain all values present in the physical memory, but those values it holds are correct, an image that is not fully correct can not be complete [34, p. 131].

3.2 Atomicity

The second constraint of Vömel and Freiling is the atomicity of a memory image. Following their definition [34, Definition 5, p.132] a memory image may be considered atomic, if and only if the created image represents the state of all processes operating on the physical memory at the same point in time [34, p.132].

More direct, this is only the case if all intrinsic processes of the surveyed system are simultaneously frozen and no further changes to the state of all memory regions occurs by those processes before the memory is read.

Hence, if the first N regions of memory have been read as they were at time point T, all subsequent memory regions have to be read in the state they have been during T, i.e. no modifications to the memory regions may have been performed by concurrently performed operations on the target [34, p.132].

3.3 Integrity

The third and last constraint introduced by Vömel and Freiling is the integrity of a memory image. A created memory image may therefore be considered to have integrity, if and only if the content of all memory regions is preserved in the same state relative to a fixed time point prior to the operations performed to recover the image [34, Definition 6, p.132].

Simply put, the memory acquisition process may not alter the memory of a target. If the memory of a target is altered, those sections altered by the acquisition process have to be recognized as altered by the acquisition utility and hence their integrity is tainted.

This also implies that a memory image may in fact provide a partial integrity.

3.4 Reproducibility

Digital Evidence used in a court of law has to have been produced in a reproducible manner. If information is extracted from something, may it be Deoxyribonucleic acid (DNA) or in this case an Information Technology system, the defendant has to be able to contract his own independent expert witness, who can re-evaluate the peoples claims about a piece of evidence.

Naturally, this is a severe issue in the case of information extracted from a systems memory, which according to Farmer and Venema ranges on their Order of Volatility (OOV) on the second most volatile rank with an expected life span of around 10 nanoseconds [7, p. 6].

While in general an image of the memory content is produced during the initial investigation, which may be permanently archived and used for the investigation, the reproducibility of the process is commonly limited.

In their 2006 paper, Sutherland et. al. approached this issue by recommending following a procedure proposed in the *"Directors and corporate advisors' guide to digital investigations and evidence"* [23], which recommends that the process of extraction is documented and observed by an independent eye-witness [26].

Such a witness may then - according to [23] - be interrogated in court, allowing the testing of the witnesses recollections of the evidence extraction, supporting the claim that the process was not tainted.

Sutherland et. al. compare this process with the interrogation of a traffic police officer during a court case handling a traffic violation observed by said officer [26, p. 67].

4 Method

As it was not possible to investigate the validity of the hypothesis on all home routers available in the

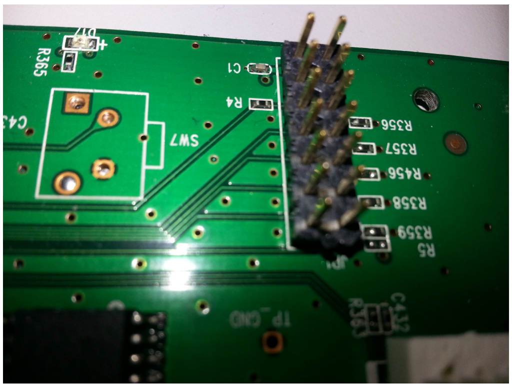

Figure 1: Equipped eJTAG header on a 1043ND.

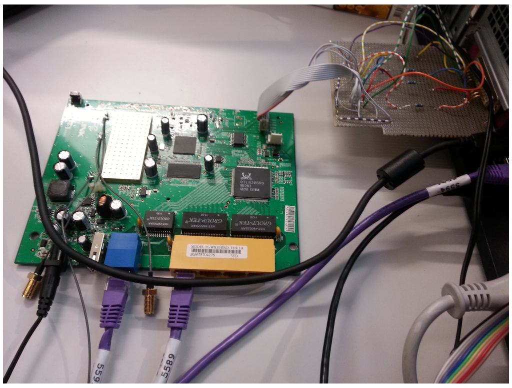

Figure 2: A DLC5 Cable is used to connect a TP-Link 1043ND with a standard PC.

market due to the vast amount of different solutions from different vendors, a single devices was chosen for a proof of concept implementation.

4.1 Investigated Device

The device that was chosen is the TP-Link 1043ND[2], a small IEEE 802.1z capable device, which also allows clients to connect via IEEE 802.1[abgn].

This device was chosen due to its easy availability in the laboratory. Furthermore, it uses a MIPS instruction set based Central Processing Unit (CPU). The Extended JTAG standard (eJTAG) Version 2.6 present in the MIPS architecture allows easy access to the system's main memory [18]. The specific central processing unit used in this model is the Atheros AR9132-BC1E which is a member of the AR71xx family.

4.2 Physical Interconnect

The eJTAG interface present on the TP-Link 1043ND was used to extract the system's running memory. To interface with the eJTAG port, the initially not accessible pin-out of the TP-Link 1043ND was equipped with a 2x7 pin row as shown in Figure 1.

This port was utilized to connect a standard PC with the 1043ND via a so called unbuffered Xilinx DLC5 Cable [36]. This cable is a fully passive connection, hence eliminates possible issues for the correctness of the obtained memory image as described in Section 3.1 due to the performance of operations in the programmable logic of an more advanced cable.

A schema of the logical connections for that cable as described in [36] can be found in Appendix A.

The cable was then interfaced with the 1043ND and a standard PC's parallel port (LTP) as seen in Figure 2, which was used to perform the data recovery tasks.

4.3 Extraction Software

To interface with the 1043ND over the eJTAG port, a special JTAG software is necessary. Due to its overall good documentation, the author chose OpenOCD[3] [20] for this project.

The used OpenOCD version, OpenOCD 0.7.0, was compiled only with support for a DLC5 parallel port interface.

In conjunction with the distribution supplied information and configuration file for AR71xx based chip-sets, OpenOCD can be utilized to extract the random access memory of a running device.

The stock OpenOCD does provide the possibility of extracting bulk chunks of memory. However, it does so by instructing the Memory Management Unit (MMU) of the CPU to deliver the desired memory contents. This is an issue for most of the constraints defined by Vömel and Freiling. Hence a method has to be implemented, that allows for the direct extraction of memory content from the device, without requiring active participation of any part of the device.

[2] `http://www.tp-link.com/en/products/details/?model=TL-WR1043ND`

[3] `http://openocd.sourceforge.net/`

As the eJTAG utilized for MIPS generally supports this [18], it should be possible to add this functionality to OpenOCD. In 2010 Timo Juhani Lindfors published a patch to an earlier version of OpenOCD[4] that allows MMU bypassed memory access. This patch can also be found in Appendix B.

As a newer OpenOCD version was utilized for the development of the method at hand, the patch had to be adjusted. The adjusted patch can be found in Appendix C.

4.4 Extraction Process

To extract the memory of the 1043ND the first step is issuing a *halt* command to the CPU with OpenOCD. This command sent via the JTAG interface immediately halts the execution of all instructions on the CPU.

In the next step OpenOCD is instructed to extract the the memory of this device in MMU bypass mode. Based on the supplied boot messages found on a serial console attached to the 1043ND and information obtained from the U-Boot boot-loader source code[5] the logical memory offset was established to be `0x80000000`. The size of the physical memory in the 1043ND is 32MB `0x2000000`, limited by the extends of the built-in chip.

With the applied physical dump patch OpenOCD will actually extract from `0xA0000000`, the physical memory location [27, p. 42ff], effectively bypassing the MMU.

Hence the command `dump_image phys img.bin 0x80000000 0x2000000` is send to OpenOCD. The memory image is extracted and then saved in the current working directory of OpenOCD with the filename supplied during the extraction process, in this case `img.bin`. The average speed during this process is around 0.65 KiB per second.

5 Validation

The proposed method has to hold up to the previously presented fundamentals of volatile memory forensic to be admissible during a forensic investigation. Hence it will be thoroughly investigated if the method complies with the presented constraints, and if it does not fully comply with those constraints, in how far the possible issues reduce the admissibility.

5.1 Correctness

The correctness of the method is highly dependent on the correctness of the OpenOCD source code. Although the original intention of OpenOCD was not forensic soundness, the initial developer aimed at a tool that "[...] never displays wrong or inaccurate information" [20, p. 38].

Even if this claim is not evaluated within the thesis documenting the initial development of OpenOCD, the author of this document assumes that this requirement is fulfilled by OpenOCD, as it would diminish the usability of a debugger if it was unfulfilled. This assumption has to be made during this research, as the thorough investigation of the source code would exceed the scope of this project.

Another important aspect of correctness, not mentioned in the 2012 paper of Vömel and Freiling is the question whether what is found in the physical memory of a device is actually correct.

Extensive research on the impact of cosmic rays on integrated circuits has been performed in the last decades [5, 14, 37, 12].

While the flipping of single bits does not necessarily pose an issue in a normal volatile memory forensic investigation, the proposed method is highly focused on the correctness of a specific, small set of bits without additional parity information. Although events that may alter those bits are considerably sparse, they may become an issue during a forensic investigation.

Specifically, Ibe et. al. could establish in viable simulations that the process of down-scaling of chips increases the cosmic ray induced soft-error frequency. Their simulations found an increase of factor 6-7 for a migration step from 130nm to 22nm [12]. With the progressing decrease in chip size these issues hence may become more problematic.

Furthermore, it has been indicated in the literature that the altitude of a location has a considerable impact on the amount of observed soft errors due to an increase in the neutron flux [30].

Although these issues can not be denied, they are not investigated in-depth in the research at hand, as this would violate the intended scope of

[4] `http://lists.berlios.de/pipermail/openocd-development/2010-November/017278.html`

[5] `http://www.denx.de/wiki/U-Boot`

the project. Further research on this matter will be advised in a later section of this document.

5.2 Completeness

The memory image created from the running device is complete, given the method itself is also correct, if all bytes present in the physical memory are also written to the memory image.

The memory image is obtained starting from the previously known offset of the physical memory. The amount of bytes read exactly equals the storage capacity in the utilized physical memory chip[6].

Hence, if the previously made assumptions hold, the proposed method is complete.

5.3 Atomicity

As previously described, the atomicity of an acquired memory snapshot is highly important [34]. Furthermore, the preserving of atomicity in the memory image acquisition process on full scale x86 based systems is a considerable challenge [22].

On the MIPS based system at hand however, this challenge is merely a minor issue. Although the extraction process is considerably slow - the extraction of one image of 32MB takes roughly 14 hours - the initial step of interrupting and freezing all processing performed on the CPU of the system ensures that no program running on that CPU is able to perform any operation on any memory area.

Therefore, even if atomicity is not ensured by the image creation process itself, but instead by a small action prior to the execution process, the atomicity of the obtained memory snapshot is preserved.

5.4 Integrity

The last constraint for a forensically viable volatile memory image is the integrity of said image. As previously described, a memory image has integrity, if and only if each of its subsets have the same state as exactly prior to the start of the acquisition process, i.e. if and only if the memory acquisition method does not alter the memory of the target system in its process.

[6]For reference: WINBOND W9425G6JH-5

Sadly OpenOCD requires a subset of the targets memory for its own processes if extended operations are performed on the system. Although according to the OpenOCD manual those operations do not include the simple memory extraction operations [1], it can not be finally be debarred, that the integrity of those memory areas has been tainted.

5.5 Reproducibility

As with most volatile memory forensics techniques, the general reproducibility of this method is highly limited. Procedural techniques as discussed in Section 3.4 may provide a sufficient documentation of the supplied method, so the reproducibility can be disregarded within the same parameters discussed for traditional volatile memory forensics methods.

Furthermore, if required in the specific case, more advanced methods could be implemented. Due to the small dimensions and the comparatively low power consumption of a device, it might be possible to conserve the device in halted state for extended time periods, if a constant power supply can be ensured. To implement such a power transfer, a utility similar to the one documented in US Patent 8,076,798 might be used [17].

5.6 Practical Image Verification

Besides the previously performed theoretical validation of the method, two other approaches for validating an obtained memory image have been tested.

The first approach is based on the work of Inoue, Adelstein and Joyce on self similarity in memory images and aims at the verification of the correctness of the proposed method. In 2011 Inoue, Adelstein and Joyce proposed the application of techniques known from biology to memory images [13]. They noticed that the presence of large amounts of self similarities within a memory image may yield that it is tainted. They researched the applicability of dotplots used in biology to illustrated self similarities to the analysis of memory images. To create such a dotplot, each page of a memory image[7] is plotted against every other page of the image. The intersection is marked, if and only if those two pages are identical and the corresponding pages do not both consist of known initialization values of memory. Inoue, Adelstein

[7]4k on MIPS32

MAC Address	Hostname
52:54:00:40:f8:7b	death
52:54:00:c1:02:83	luggage
52:54:00:74:68:4d	poons
52:54:00:5c:af:ca	ridcully
52:54:00:6f:b0:32	rincewind
52:54:00:90:58:05	stibbons
52:54:00:00:ad:04	twoflower
52:54:00:d5:4e:33	vimmes

Table 1: List of used hosts with their associated MAC addresses.

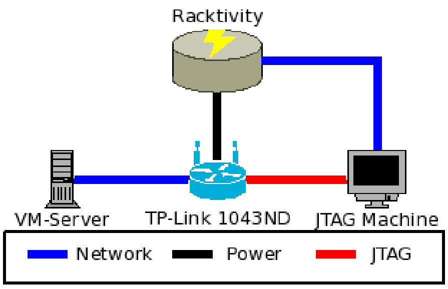

Figure 4: Schematic representation of the setup used for testing the proposed method.

and Joyce found that this method from biology is applicable to memory images and allows for the detection of tainted image acquisition techniques [13].

This technique proposed by Inoue, Adelstein and Joyce has been applied to memory images obtained with the method proposed by the author. One of the obtained dotplots can be seen in Figure 3. During the creation of the dotplot, all pages that contained only values related to memory initialization have been ignored. In this case these were pages exclusively containing either $0x00$, $0xFF$, $0x55$ or $0xAA$.

The presented dotplot clearly shows no significant self similarity. This supports the conclusion of the theoretical evaluation that the proposed method is correct in a forensic context.

The second practical validation approach is related to the theoretical validation of the integrity of the proposed method. The higher the integrity of the method, the more similarity should exist between two images, subsequently taken from the same source.

Ideally both images should be identical bitwise.

The author performed those subsequent memory extractions two times. Both times the two images extracted within one process were bitwise identical.

5.7 Summary

After theoretically evaluating the previously postulated requirements for a forensically sound memory acquisition technique, the proposed method may be considered sufficiently viable in a forensic context.

A short practical evaluation further strengthens the theoretically obtained conclusions. Hence the author assumes the proposed method viable for a forensic memory extraction of volatile memory from home routers.

Therefore, the practical testing of the previously postulated hypothesis regarding the extractability of DHCP lease file information is the next step.

6 Empirical Verification

To empirically test the practical feasibility of the proposed method, the already presented TP-Link 1043ND was incorporated in an automated test setup. The setup would transparently simulate various scenarios and then extract the device memory with the proposed method. The extracted memory image can then be investigated.

As the simulated scenario is known prior to the investigation of the extracted memory images, the results of the investigation can be compared to the actual events, providing a metric of efficiency for this method.

6.1 Experiment Setup

The setup consisted of four main elements as shown in Figure 4. The core of the setup was the TP-Link 1043ND. A virtual machine server was connected to the internal network ports of the router, providing the virtual machines mimicking the network clients of the home router. A secondary system was connected to the JTAG port of the 1043ND and finally a Racktivity Remote Power Switch Unit[8] provided the capability of remotely switching the 1043ND on and off, i.e. resetting it to the initial state.

[8]http://www.racktivity.com/

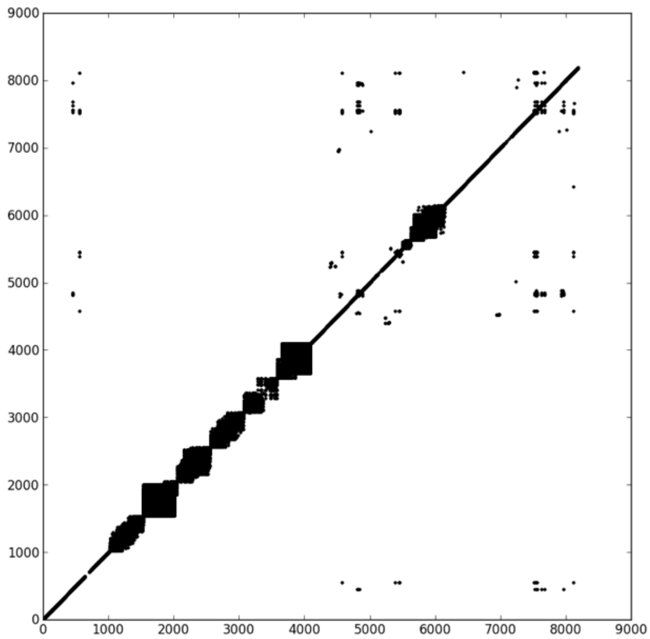

Figure 3: Dotplot showing self-similarity between pages in a memory image obtained by the author. The axis show the index of the corresponding pages.

The initial state in this case means that everything is powered off.

6.2 Data Acquisition

The experiments were then conducted using the Python script supplied in Appendix D. For each iteration of a simulated scenario, the whole setup would first be reset. This means that all virtual machines are stopped, and the home router was powered off for 120 seconds to erase all possible artifacts in memory that may survive a short cold or even hot reset of the device. The time-frame of 120 seconds was chosen, as research on cold boot attacks by Halderman et. al. suggests that the bits stored in modern SRAM chips decay to a state where non of the initial data is left after 50 seconds on standard operation temperature[9] [11]. Hence the power-off time of 120 seconds should ensure that no patterns are left in the memory after a reset of the device.

After the initial setup reset, the 1043ND would be powered on. Based on the settings for each scenario the virtual machines would then be powered on and off at will.

In total eight different scenarios were created for the purpose of this experiment. The timelines of the actual events can be found in Appendix E. These timelines are all relative to the initial power-on of the device. A rough outline of the conducted experiments can be found in Table 2

7 Results

During the investigation of the memory images from the 1043ND, it was discovered that the device seems to hold the lease file not on a memory filesystem as expected, but instead directly in the DHCP process' memory.

An example excerpt can be found in Appendix I. The first 48bit on line $0x01f691f0$ e.g. show the beginning of the entry for one host, starting with the recognizable vendor id part of KVM hosts' MAC addresses, *52:54:00:XX:XX:XX*.

When compared with the MAC addresses used in the experiment as listed in Table 1, it becomes

[9] $25.5°C$ to $44.1°C$ in the experiments of Halderman et. al.

Scenario	Description
adv-test-1-4	boot 1 host, shutdown, wait 4h, dump memory
adv-test-1-8	boot 1 host, shutdown, wait 8h, dump memory
adv-test-8-4	boot 8 hosts, shutdown, wait 4h, dump memory
adv-test-8-8	boot 8 hosts, shutdown, wait 8h, dump memory
plain-test-4	boot 4 hosts, dump memory
plain-test-8	boot 8 hosts, dump memory
complex	boot 3 hosts, wait 1.25h, boot 3 hosts, shutdown 2 hosts, wait 12h, dump memory

Table 2: Overview of the simulated scenarios.

even more obvious, it can be observed that the data found in the presented excerpt actually corresponds to the MAC addresses of hosts used in the experiments.

Although an extraction of the information would be possible, another angle of approach was found. The 1043ND firmware also logs the DHCP server related events to the system console. The content of said console can be found as plain ASCII values within the created memory dumps. An example of those loglines can be found in Appendix H.

As plain strings can be handled more easily, the syslog information was used for the timeline estimation.

Furthermore, issues with the global time correlation of the events taking place on the device were discovered. The model at hand seems to be unable to synchronize its local time with sources from the Internet. Hence no relation between real time and device relative time is easily possible.

Although tools like e.g. volatility[10] support the extraction of the uptime, which could be used to establish a connection between the real and the device time, support for MIPS based memory dumps is not present.

To preserve the scope of this project, a different approach was taken, to estimate the overall device run time. As previously mentioned, the device system log information can be found in memory. To estimate the uptime, the oldest as well as the newest entry in the extracted syslog are used. The difference between the two points in time is then considered to be the uptime of the device. In conjunction with the known external time of the device, halt events with known device relative time can then be correlated to real times.

The extraction of information was conducted with a Python script designed specifically for this purpose. This script can be found in Appendix G. The script creates visual and textual representations of the extracted timelines. Tables showing those timelines can be found in Appendix F.

From the created timelines, four different metrics were extracted.

1. Correctly detected clients.
2. Correctly detected join-time.
3. Amount of MACs in DHCPD memory.
4. Amount of false positives.

The detection rate of all clients is the count of all hosts that were used in a scenario and have been detected in the image.

The detection accuracy of device join-time is the amount of hosts where the detected network join-time corresponds to the host's boot time. Due to the boot process, a timeframe of 120 seconds is accepted as slack.

The amount of false positives consists of all hosts that have been determined to be present from the memory dump, although they have not been present, and the list of host in memory are all hosts for which MAC addresses have been found in the memory region suspected to hold the DHCP lease file. The latter operation has been performed manually.

7.1 Extracted Information

The two simple scenarios show a very good detection rate of the connected hosts. All hosts used in the experiment could also be identified in the associated memory dump using the developed tooling.

The four advanced scenarios show far more different results. While the presence of a single host with only one active host overall was reliably detected even after four and seven hours, the detection rate in the case of seven hosts being present was considerably low.

[10] `http://code.google.com/p/volatility/`

Scenario	Detection Rate	Accuracy	Hosts In-Memory	False Positives	Total Hosts
adv-test-1-4	1	1	1	0	1
adv-test-1-8	1	1	1	0	1
adv-test-8-4	2	2	8	0	8
adv-test-8-8	2	2	8	0	8
plain-test-4	4	4	4	0	4
plain-test-8	8	8	8	0	8
complex	6	3	6	0	6

Table 3: Results for the seven scenarios in the three different metrics.

This may however be related to the choice of the system log messages as data-source. A manual investigation of the corresponding memory images could produce the MAC addresses of all hosts utilized in this experiment from the memory. As expected, based on the previous observation all of them resided in coherent block of the memory dump, which is therefore assumed to belong to the DHCP server process on the home router.

The complex test, in which a more complicated set of events was created, could be identified in so far, that there was no occurrence of a host being determined to be present during a timeframe when it was in fact not present, i.e. no false positives occurred.

Furthermore, an estimate of when a host was present could be established.

To summarize the results, it can be held up that timelines could be created for various scenarios. Although some weaknesses exist on older data, these may be avoidable by utilizing more aspects of the memory resource.

8 Conclusion

The obtained results show that the proposed method is viable. The method did not - very importantly - show false positives. In all cases all relevant hosts could be identified, either with the implemented method or with easily possible extensions. Further more the data extracted from the memory dumps allows the creation of admissible timelines within reasonable bounds.

Hence the initially postulated hypothesis, that information extracted from the main memory of a home router should allow the creation of a timeline showing the association and disassociation of devices in a forensically viable manner, can be considered to be confirmed.

The proposed method is viable and should be implementable for all available home routing devices that have an accessible eJTAG or JTAG port.

Although the base hypothesis has been confirmed, further research and engineering work is necessary to produce a market ready solution that allows the collection of forensically sound evidence from home routers.

8.1 Further Work

Many different chip-sets and firmware versions exist on the market. The investigation of these devices, how to interface with their specific JTAG ports - ideally in a soldering free manner[11] - and the creation of dedicated, tailored extraction utilities for all of these devices is probably the most important step.

This creation process also includes the investigation of different lease file formats and their extractability and informational value. As indicated in the results section, an additional incorporation of the actual lease file in the DHCP servers memory may have significantly increased the reliability and accuracy of the proposed method.

Furthermore, different angles of extractable information should be the matter of further research, as the overall impact of home router memory forensics could be highly increased by incorporating additional sources of information.

Home routing devices conduct various operations that may be of forensic interest and might allow a significant leap during the investigation of a case. Many devices do not only provide networking capabilities, but also further operations like Dynamic Name System (DNS) caching-resolver services or printing services.

These devices also handle all network traffic of a specific network segment. Due to the implemen-

[11] A technique discussed on the Internet in 2009 may be viable here: `http://www.usbjtag.com/vbforum/showpost.php?p=18571`

tation specifics of NAT, the devices have to keep state for all connections performed via them [24]. In their 2011 paper, Beverly et. al. provided evidence that those network structures may be extracted from memory images, even hours after they actually happened [2].

An investigation like this could allow the determination of connections performed by devices that already have been shut down or removed from the vicinity before the investigation.

8.2 Defense Mechanisms

The matter of defending against this method is important, as this method might also be used by unlawful regimes against legitimate interests.

A defense against the presented methods is complicated. The DHCP lease information has to be stored somewhere. While dedicated systems with strict on-disk lease files and sophisticated measures against key extraction from memory via cold-boot attacks as described by Halderman et. al. [11] might be viable for a very small set of technologically advanced subjects, this does not help against this issue for most average customers.

A defense system for the masses could most probably only be implemented, if version 6 of the Internet Protocol, which allows a stateless address auto configuration, is used in conjunction with IPv6 privacy extensions, which masks the specific device identifier of a certain system.

That way no state has to be kept on a device, hence the target for possibly extractable information is highly reduced.

The most promising defense method is, however, the physical removal of JTAG access or disabling thereoff in the chip.

Acknowledgments

Thomas Roth - For his support on Architectures.
Andreas Schuster - For feedback and supervision.

References

[1] OpenOCD User's Guide 0.8.0-dev, Section 11.3, 2013. `http://openocd.sourceforge.net/doc/html/CPU-Configuration.html`, accessed: Tue Jun 25 19:17:33 CEST 2013.

[2] Robert Beverly, Simson Garfinkel, and Greg Cardwell. Forensic carving of network packets and associated data structures. *Digital Investigation*, 8:S78–S89, 2011.

[3] Roberto Bez, Emilio Camerlenghi, Alberto Modelli, and Angelo Visconti. Introduction to flash memory. *Proceedings of the IEEE*, 91(4):489–502, 2003.

[4] Ing Breeuwsma et al. Forensic imaging of embedded systems using JTAG (boundary-scan). *Digital Investigation*, 3(1):32–42, 2006.

[5] V Degalahal, R Ramanarayanan, N Vijaykrishnan, Y Xie, and MJ Irwin. The effect of threshold voltages on the soft error rate [memory and logic circuits]. In *5th International Symposium on Quality Electronic Design, 2004. Proceedings.*, pages 503–508. IEEE, 2004.

[6] R. Droms. Dynamic Host Configuration Protocol. RFC 2131 (Draft Standard), March 1997. Updated by RFCs 3396, 4361, 5494, 6842.

[7] Dan Farmer and Wietse Venema. *Forensic Discovery*. Addison-Wesley Reading, 2005.

[8] Robert Swope Fleming. *The end of the Intel age*. PhD thesis, Massachusetts Institute of Technology, 2011.

[9] Howard Frazier. The 802.3z gigabit Ethernet standard. *IEEE Network*, 12(3):6–7, 1998.

[10] Steve B Furber. *ARM system Architecture*. Addison-Wesley Longman Publishing Co., Inc., 1996.

[11] J Alex Halderman, Seth D Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A Calandrino, Ariel J Feldman, Jacob Appelbaum, and Edward W Felten. Lest we remember: cold-boot attacks on encryption keys. *Communications of the ACM*, 52(5):91–98, 2009.

[12] Eishi Ibe, Hitoshi Taniguchi, Yasuo Yahagi, K-i Shimbo, and Tadanobu Toba. Impact of scaling on neutron-induced soft error in SRAMs from a 250 nm to a 22 nm design rule. *IEEE Transactions on Electron Devices*, 57(7):1527–1538, 2010.

[13] Hajime Inoue, Frank Adelstein, and Robert A Joyce. Visualization in testing a volatile memory forensic tool. *Digital Investigation*, 8:S42–S51, 2011.

[14] Tanay Karnik and Peter Hazucha. Characterization of soft errors caused by single event upsets in CMOS processes. *IEEE Transactions on Dependable and Secure Computing*, 1(2):128–143, 2004.

[15] Zhongli Liu, Yinjie Chen, Wei Yu, and Xinwen Fu. Generic network forensic data acquisition from household and small business wireless routers. In *2010 IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM)*, pages 1–6. IEEE, 2010.

[16] Colin M Maunder and Rodham E Tulloss. *The test access port and boundary-scan architecture*. IEEE Computer Society Press Los Alamitos/Washington, DC, 1990.

[17] Dean L Mehler and James P Wiebe. Uninterruptible a/c power supply transfer unit, December 13, 2011. US Patent 8,076,798.

[18] MIPS Technologies Inc. *MIPS32 4KE™Processor Core Family Software User's Manual*. MIPS Technologies Inc., 2002.

[19] Jon Postel and Joyce K Reynolds. Standard for the transmission of IP datagrams over IEEE 802 networks. 1988.

[20] Dominic Rath. Open On-Chip Debugger, 2008.

[21] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address Allocation for Private Internets. RFC 1918 (Best Current Practice), February 1996. Updated by RFC 6761.

[22] Bradley Schatz. BodySnatcher: Towards reliable volatile memory acquisition by software. *Digital Investigation*, 4:126–134, 2007.

[23] Peter Sommer. Directors and corporate advisors' guide to Digital Investigations and evidence. 2005.

[24] P. Srisuresh and M. Holdrege. IP Network Address Translator (NAT) Terminology and Considerations. RFC 2663 (Informational), August 1999.

[25] Ryszard Struzak. Broadband internet in EU countries: limits to growth. *IEEE Communications Magazine*, 48(4):52–57, 2010.

[26] Iain Sutherland, Jon Evans, Theodore Tryfonas, and Andrew Blyth. Acquiring volatile operating system data tools and techniques. *ACM SIGOPS Operating Systems Review*, 42(3):65–73, 2008.

[27] Dominic Sweetman. *See MIPS run*. Morgan Kaufmann, 2010.

[28] Patryk Szewczyk. ADSL Router Forensics Part 1: An introduction to a new source of electronic evidence. In *Australian Digital Forensics Conference*, page 13, 2007.

[29] Patryk Szewczyk. ADSL Router Forensics Part 2: Acquiring Evidence. 2009.

[30] Yoshiharu Tosaka, Ryozo Takasu, Taiki Uemura, Hideo Ehara, Hideya Matsuyama, Shigeo Satoh, Atsushi Kawai, and Masahiko Hayashi. Simultaneous measurement of soft error rate of 90 nm cmos sram and cosmic ray neutron spectra at the summit of mauna kea. In *IEEE International Reliability Physics Symposium, 2008. IRPS 2008.*, pages 727–728. IEEE, 2008.

[31] Onno Van Eijk and Mark Roeloffs. Forensic acquisition and analysis of the Random Access Memory of TomTom GPS navigation systems. *Digital Investigation*, 6(3):179–188, 2010.

[32] Richard Van Nee, Geert Awater, Masahiro Morikura, Hitoshi Takanashi, Mark Webster, and Karen W Halford. New high-rate wireless LAN standards. *IEEE Communications Magazine*, 37(12):82–88, 1999.

[33] Dimitris Vassis, George Kormentzas, Angelos Rouskas, and Ilias Maglogiannis. The IEEE 802.11g standard for high data rate WLANs. *IEEE Network*, 19(3):21–26, 2005.

[34] Stefan Vömel and Felix C Freiling. Correctness, atomicity, and integrity: Defining criteria for forensically-sound memory acquisition. *Digital Investigation*, 9(2):125–137, 2012.

[35] Yang Xiao. IEEE 802.11 n: enhancements for higher throughput in wireless LANs. *IEEE Wireless Communications*, 12(6):82–91, 2005.

[36] Xilinx Inc. JTAG Programmer Guide, Appendix B, 1999.

[37] James F Ziegler, HW Curtis, HP Muhlfeld, CJ Montrose, B Chin, M Nicewicz, CA Russell, WY Wang, LB Freeman, P Hosier, et al. IBM experiments in soft fails in computer electronics (1978–1994). *IBM journal of research and development*, 40(1):3–18, 1996.

A DLC5 Xilinx Cable Simple Version

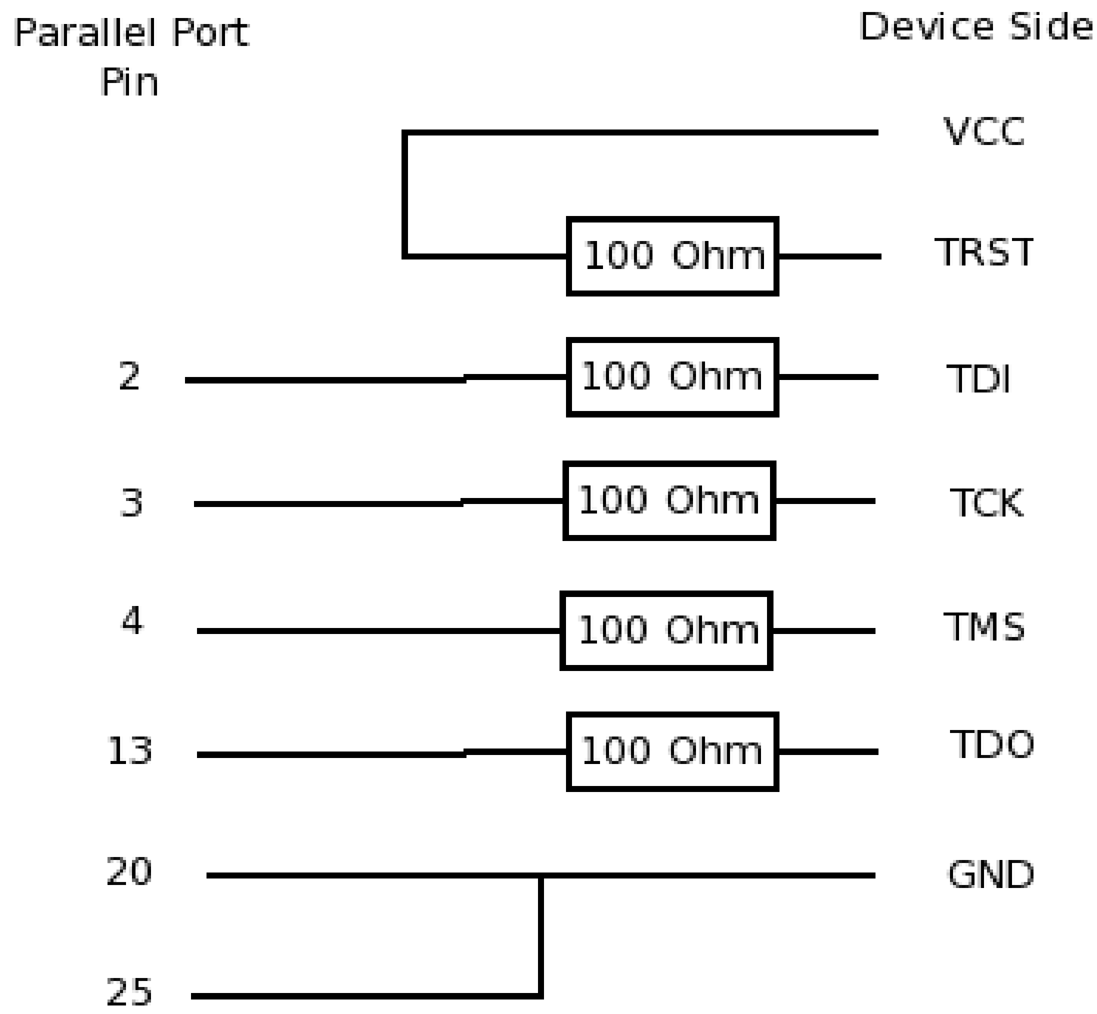

Figure 5: Schematics of the used simplified DLC5 Cable. The graphic is based on an image by the user “RealOpty” in the OpenWRT Wiki (`http://wiki.openwrt.org/doc/hardware/port.jtag.cable.unbuffered`) viewed on Mon Jul 1 21:15:41 CEST 2013.

B Unmodified Version of the OpenOCD DMA Patch

```
diff --git a/doc/openocd.texi b/doc/openocd.texi
index 70d789a..1b75f94 100644
--- a/doc/openocd.texi
+++ b/doc/openocd.texi
@@ -5731,9 +5731,12 @@ Otherwise, or if the optional @var{phys} flag is specified,
 @cindex image dumping

 @anchor{dump_image}
- at deffn Command {dump_image} filename address size
-Dump @var{size} bytes of target memory starting at @var{address} to the
-binary file named @var{filename}.
+ at deffn Command {dump_image} [phys] filename address size
+Dump @var{size} bytes of target memory starting at @var{address} to
+the binary file named @var{filename}. When the current target has an
+MMU which is present and active, @var{addr} is interpreted as a
+virtual address.  Otherwise, or if the optional @var{phys} flag is
+specified, @var{addr} is interpreted as a physical address.
 @end deffn

 @deffn Command {fast_load}
diff --git a/src/target/target.c b/src/target/target.c
index 93efa76..5cc1e6a 100644
--- a/src/target/target.c
+++ b/src/target/target.c
@@ -1393,12 +1393,22 @@ int target_write_buffer(struct target *target, uint32_t address, uint32_t size,
  * mode respectively, otherwise data is handled as quickly as
  * possible
  */
-int target_read_buffer(struct target *target, uint32_t address, uint32_t size, uint8_t *buffer)
+static int target_read_buffer2(struct target *target, uint32_t address, uint32_t size, uint8_t *buffer, bool physical)
 {
  int retval;
  LOG_DEBUG("reading buffer of %i byte at 0x%8.8x",
        (int)size, (unsigned)address);

+ int (*read_fn)(struct target *target,
+     uint32_t address, uint32_t size, uint32_t count, uint8_t *buffer);
+ if (physical)
+ {
+   read_fn=target_read_phys_memory;
+ } else
+ {
+   read_fn=target_read_memory;
+ }
+
  if (!target_was_examined(target))
  {
    LOG_ERROR("Target not examined yet");
@@ -1420,7 +1430,7 @@ int target_read_buffer(struct target *target, uint32_t address, uint32_t size, u

  if (((address % 2) == 0) && (size == 2))
  {
-   return target_read_memory(target, address, 2, 1, buffer);
+   return read_fn(target, address, 2, 1, buffer);
  }

  /* handle unaligned head bytes */
@@ -1431,7 +1441,7 @@ int target_read_buffer(struct target *target, uint32_t address, uint32_t size, u
    if (unaligned > size)
      unaligned = size;

-   if ((retval = target_read_memory(target, address, 1, unaligned, buffer)) != ERROR_OK)
```

```
+    if ((retval = read_fn(target, address, 1, unaligned, buffer)) != ERROR_OK)
       return retval;

     buffer += unaligned;
@@ -1444,7 +1454,7 @@ int target_read_buffer(struct target *target, uint32_t address, uint32_t size, u
   {
     int aligned = size - (size % 4);

-    if ((retval = target_read_memory(target, address, 4, aligned / 4, buffer)) != ERROR_OK)
+    if ((retval = read_fn(target, address, 4, aligned / 4, buffer)) != ERROR_OK)
       return retval;

     buffer += aligned;
@@ -1456,7 +1466,7 @@ int target_read_buffer(struct target *target, uint32_t address, uint32_t size, u
   if(size >=2)
   {
     int aligned = size - (size%2);
-    retval = target_read_memory(target, address, 2, aligned / 2, buffer);
+    retval = read_fn(target, address, 2, aligned / 2, buffer);
     if (retval != ERROR_OK)
       return retval;

@@ -1467,13 +1477,18 @@ int target_read_buffer(struct target *target, uint32_t address, uint32_t size, u
   /* handle tail writes of less than 4 bytes */
   if (size > 0)
   {
-    if ((retval = target_read_memory(target, address, 1, size, buffer)) != ERROR_OK)
+    if ((retval = read_fn(target, address, 1, size, buffer)) != ERROR_OK)
       return retval;
   }

   return ERROR_OK;
 }

+int target_read_buffer(struct target *target, uint32_t address, uint32_t size, uint8_t *buffer)
+{
+ target_read_buffer2(target, address, size, buffer, false);
+}
+
 int target_checksum_memory(struct target *target, uint32_t address, uint32_t size, uint32_t* crc)
 {
   uint8_t *buffer;
@@ -2605,6 +2620,12 @@ COMMAND_HANDLER(handle_dump_image_command)
   struct duration bench;
   struct target *target = get_current_target(CMD_CTX);

+ bool physical=strcmp(CMD_ARGV[0], "phys")==0;
+ if (physical)
+ {
+   CMD_ARGC--;
+   CMD_ARGV++;
+ }
   if (CMD_ARGC != 3)
     return ERROR_COMMAND_SYNTAX_ERROR;

@@ -2622,7 +2643,7 @@ COMMAND_HANDLER(handle_dump_image_command)
   {
     size_t size_written;
     uint32_t this_run_size = (size > 560) ? 560 : size;
-    retval = target_read_buffer(target, address, this_run_size, buffer);
+    retval = target_read_buffer2(target, address, this_run_size, buffer, physical);
     if (retval != ERROR_OK)
     {
       break;
```

```
@@ -5305,7 +5326,7 @@ static const struct command_registration target_exec_command_handlers[]
= {
 		.name = "dump_image",
 		.handler = handle_dump_image_command,
 		.mode = COMMAND_EXEC,
-		.usage = "filename address size",
+		.usage = "['phys'] filename address size",
 	},
 	{
 		.name = "verify_image",
--
1.7.2.3
```

C Modified Version of the OpenOCD DMA Patch

```
diff --git a/openocd-0.7.0/src/target/target.c b/openocd-0.7.0/src/target/target.c
index ed1a2cc..870d757 100644
--- a/openocd-0.7.0/src/target/target.c
+++ b/openocd-0.7.0/src/target/target.c
@@ -1758,11 +1758,22 @@ static int target_write_buffer_default(struct target *target, uint32_t address,
   * mode respectively, otherwise data is handled as quickly as
   * possible
   */
-int target_read_buffer(struct target *target, uint32_t address, uint32_t size, uint8_t *buffer)
+static int target_read_buffer2(struct target *target, uint32_t address, uint32_t size, uint8_t *buffer, bool physical)
 {
  LOG_DEBUG("reading buffer of %i byte at 0x%8.8x",
        (int)size, (unsigned)address);

+// int (*read_fn)(struct target *target,
+//     uint32_t address, uint32_t size, uint32_t count, uint8_t *buffer);
+ if (physical)
+ {
+   read_fn=target_read_phys_memory;
+ } else
+ {
+   read_fn=target_read_memory;
+ }
+
+
  if (!target_was_examined(target)) {
    LOG_ERROR("Target not examined yet");
    return ERROR_FAIL;
@@ -1787,7 +1798,7 @@ static int target_read_buffer_default(struct target *target, uint32_t address, u
  int retval = ERROR_OK;

  if (((address % 2) == 0) && (size == 2))
-   return target_read_memory(target, address, 2, 1, buffer);
+   return read_fn(target, address, 2, 1, buffer);

  /* handle unaligned head bytes */
  if (address % 4) {
@@ -1796,7 +1807,7 @@ static int target_read_buffer_default(struct target *target, uint32_t address, u
    if (unaligned > size)
      unaligned = size;

-   retval = target_read_memory(target, address, 1, unaligned, buffer);
+   retval = read_fn(target, address, 1, unaligned, buffer);
    if (retval != ERROR_OK)
      return retval;

@@ -1810,6 +1821,7 @@ static int target_read_buffer_default(struct target *target, uint32_t address, u
    int aligned = size - (size % 4);

    retval = target_read_memory(target, address, 4, aligned / 4, buffer);
+   retval = read_fn(target, address, 4, aligned / 4, buffer);
    if (retval != ERROR_OK)
      return retval;

@@ -1821,7 +1833,8 @@ static int target_read_buffer_default(struct target *target, uint32_t address, u
  /*prevent byte access when possible (avoid AHB access limitations in some cases)*/
  if (size  >= 2) {
    int aligned = size - (size % 2);
-   retval = target_read_memory(target, address, 2, aligned / 2, buffer);
+   retval = read_fn(target, address, 2, aligned / 2, buffer);
```

```
+
     if (retval != ERROR_OK)
       return retval;

@@ -1831,7 +1844,7 @@ static int target_read_buffer_default(struct target *target, uint32_t address, u
   }
   /* handle tail writes of less than 4 bytes */
   if (size > 0) {
-    retval = target_read_memory(target, address, 1, size, buffer);
+    retval = read_fn(target, address, 1, size, buffer);
     if (retval != ERROR_OK)
       return retval;
   }
@@ -1839,6 +1852,11 @@ static int target_read_buffer_default(struct target *target, uint32_t address, u
   return ERROR_OK;
 }

+int target_read_buffer(struct target *target, uint32_t address, uint32_t size, uint8_t *buffer)
+{
+ return target_read_buffer2(target, address, size, buffer, false);
+}
+
 int target_checksum_memory(struct target *target, uint32_t address, uint32_t size, uint32_t* crc)
 {
  uint8_t *buffer;
@@ -2888,6 +2906,14 @@ COMMAND_HANDLER(handle_dump_image_command)
  struct duration bench;
  struct target *target = get_current_target(CMD_CTX);

+ bool physical=strcmp(CMD_ARGV[0], "phys")==0;
+ if (physical)
+ {
+   CMD_ARGC--;
+   CMD_ARGV++;
+ }
+
+
  if (CMD_ARGC != 3)
    return ERROR_COMMAND_SYNTAX_ERROR;

@@ -2910,7 +2936,7 @@ COMMAND_HANDLER(handle_dump_image_command)
  while (size > 0) {
    size_t size_written;
    uint32_t this_run_size = (size > buf_size) ? buf_size : size;
-   retval = target_read_buffer(target, address, this_run_size, buffer);
+   retval = target_read_buffer2(target, address, this_run_size, buffer, physical);
    if (retval != ERROR_OK)
      break;

@@ -5650,7 +5676,7 @@ static const struct command_registration target_exec_command_handlers[] = {
    .name = "dump_image",
    .handler = handle_dump_image_command,
    .mode = COMMAND_EXEC,
-   .usage = "filename address size",
+   .usage = "['phys'] filename address size",
  },
  {
    .name = "verify_image",
diff --git a/openocd-0.7.0/src/target/target.h b/openocd-0.7.0/src/target/target.h
index e6b931d..00d6925 100644
--- a/openocd-0.7.0/src/target/target.h
+++ b/openocd-0.7.0/src/target/target.h
@@ -587,4 +587,8 @@ void target_handle_event(struct target *t, enum target_event e);

 extern bool get_target_reset_nag(void);
```

```

+/* adding definition for read_fn */
+int (*read_fn)(struct target *target, uint32_t address, uint32_t size, uint32_t count,
uint8_t *buffer);
+
+
 #endif /* TARGET_H */
```

D Utility for Automated Method Tests

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import warnings
warnings.filterwarnings("ignore")

import sys
import os
import re
import telnetlib
import libvirt
import time
import datetime
import subprocess
#import tempfile
#from optparse import OptionParser

libvirt_domains = ['ridcully', 'vimmes', 'death', 'rincewind', 'poons', 'luggage', 'twoflower', 'stibbons']
ram_offset = 0x80000000
ram_size = 0x2000000
#ram_size = 0x2000
debug = False

# global log directive
def log(str_f, line):
  now = datetime.datetime.now()
  f = open('./logs/'+str_f, "a+")
  f.write(str(now) + ": " + line + "\n")
  print str_f+": " + str(now) + ": " + line

def print_percent(str_file):
  f_max = float(ram_size / 1024)
  f_cur = 0
  cur_pref = 0
  stale_counter = 0

  print "processing "+str_file+" : "
  while f_cur < f_max:
    proc = subprocess.Popen(["du ~/ocd/"+str_file], stdout=subprocess.PIPE, shell=True)
    (out, err) = proc.communicate()

    f_cur = float(out.strip().split()[0])

    f_per = (f_cur/f_max * 100)

    sys.stdout.flush()
    sys.stdout.write("[ "+"#"*int(f_per * 0.78)+" "*int((100-int(f_per-1))*0.78) +" ] "+str(round(f_per,2))+"%\r")
    sys.stdout.flush()
    time.sleep(1)
    if cur_pref == f_cur:
      stale_counter = stale_counter + 1
    elif cur_pref < f_cur:
      stale_counter = 0
    if stale_counter > 60:
      f_cur = f_max
  print ""

def print_percent_sleep(s_time):
  f_max = float(s_time)
  f_cur = 0.0
  while f_cur <= f_max:
    f_per = (f_cur/f_max * 100)
    sys.stdout.flush()
    sys.stdout.write("[ "+"#"*int(f_per * 0.78)+" "*int((100-int(f_per-1))*0.78) +" ]  cur: "+str(f_cur)+"s ("+str(round(f_per,2))+"%)\r")
```

```
    sys.stdout.flush()
    time.sleep(1)
    f_cur = f_cur + 1
  print ""


# racktivity controll section
def set_port(str_host, str_port, str_user, str_pass, str_unit, int_port, int_state):
  ret = "Port "+str(int_port)+" on "+str_unit+" set to "+str(int_state)+": "
  tn = telnetlib.Telnet(str_host, str_port)
  tn.read_until("Login: ")
  tn.write(str_user + "\n")
  tn.read_until("Password: ")
  tn.write(str_pass + "\n")

  tn.read_until("Login successful.")

  tn.write("\n")
  tn.read_until("Login: ")
  tn.write(str_user + "\n")
  tn.read_until("Password: ")
  tn.write(str_pass + "\n")

  tn.read_until("PROMPT>")


  tn.write("SET "+str(str_unit)+" PORTSTAT "+str(int_port)+" "+str(int_state)+"\n")
  tn.read_until("\n")
  ret += tn.read_until("\n").strip()
  tn.read_until("PROMPT>")
  tn.write("LOGOUT\n")

  tn.read_until("Command OK:")
  tn.close()
  return ret

def port_on(int_port):
  return set_port('192.168.23.2', '2001', 'USERNAME', 'PASSWORD', 'P1', int_port, 1)

def port_off(int_port):
  return set_port('192.168.23.2', '2001', 'USERNAME', 'PASSWORD', 'P1', int_port, 0)

# memory aquisition
def get_mem_generic(str_host, str_port, str_file_prfx, hex_offset, hex_size):
  ret = ""

  str_file = str_file_prfx+"-"+str(hex(hex_offset)).strip('L')+"-"+str(hex(hex_size)).strip('L')+".bin"

  tn = telnetlib.Telnet(str_host, str_port)
  tn.read_until(">")
  tn.write("reset\n")
  tn.read_until(">")
  tn.write("halt\n")
  tn.read_until(">")
  tn.write("dump_image phys "+str_file+" "+str(hex(hex_offset)).strip('L')+"
  "+str(hex(hex_size)).strip('L')+"\n")
  print_percent(str_file)
  tn.read_until("\n")
  ret = tn.read_until("\n")
  tn.read_until(">")
  tn.write("exit\n")
  tn.close()

  return ret.strip() + " to: "+str_file

def get_mem(str_file_prfx, hex_offset, hex_size):
  return get_mem_generic('127.0.0.1', '4444', str_file_prfx, hex_offset, hex_size)

# libvirt controlls
```

```
def libvirt_test():
  conn=libvirt.open("qemu+tcp://192.168.23.1/system")
  print conn.numOfDomains()
  print conn.listDefinedDomains()
  print conn.listDomainsID()
  conn.close()

def lv_start_n(number):
  conn=libvirt.open("qemu+tcp://192.168.23.1/system")
  vms = conn.listDefinedDomains()
  ret = []
  for idx in range(0,number):
    d = conn.lookupByName(vms[idx])
    ret.append(vms[idx])
    d.create()
  return ret

def lv_stop_all():
  conn=libvirt.open("qemu+tcp://192.168.23.1/system")
  ret = "Stopped "+str(conn.numOfDomains())+" domains."
  ids = conn.listDomainsID()
  for dom in ids:
    d = conn.lookupByID(dom)
    d.destroy()
  conn.close()
  return ret

def lv_stop_host(host):
  conn=libvirt.open("qemu+tcp://192.168.23.1/system")
  ret = "Stopped "+host
  d = conn.lookupByName(host)
  d.destroy()
  conn.close()
  return ret

# utility functions
def reset():
  ll = "reseting setup"
  log("generic.txt", ll)
  ll = port_off(1)
  log("generic.txt", ll)
  ll = lv_stop_all()
  log("generic.txt", ll)
  if not debug:
    log("generic.txt", "Sleeping for 120s")
    print_percent_sleep(120)
  else:
    log("generic.txt", "Sleeping for 10s")
    print_percent_sleep(10)

def test_plain(int_hosts):
  ll = port_on(1)
  log("plain-test-"+str(int_hosts)+"-hosts.txt", ll)

  if not debug:
    hosts = lv_start_n(int_hosts)
    ll = "started: "+str(hosts)
    log("plain-test-"+str(int_hosts)+"-hosts.txt", ll)

    log("plain-test-"+str(int_hosts)+"-hosts.txt", "Sleeping for 460s")
    print_percent_sleep(460)
  else:
    log("plain-test-"+str(int_hosts)+"-hosts.txt", "Sleeping for 20s")
    print_percent_sleep(20)

  ll = get_mem("plain-test-"+str(int_hosts)+"-hosts", ram_offset, ram_size)
  log("plain-test-"+str(int_hosts)+"-hosts.txt", ll)

def test_adv(int_hosts, int_dist):
  ll = port_on(1)
```

```
  log("adv-test-"+str(int_hosts)+"-"+str(int_dist)+"-hosts.txt", ll)

  if not debug:
    hosts = lv_start_n(int_hosts)
    ll = "started: "+str(hosts)
    log("adv-test-"+str(int_hosts)+"-"+str(int_dist)+"-hosts.txt", ll)

    log("adv-test-"+str(int_hosts)+"-"+str(int_dist)+"-hosts.txt", "Sleeping for 460s")
    print_percent_sleep(460)

    ll = lv_stop_all()
    log("adv-test-"+str(int_hosts)+"-"+str(int_dist)+"-hosts.txt", ll)

    sleep = int_dist * 3600
    log("adv-test-"+str(int_hosts)+"-"+str(int_dist)+"-hosts.txt", "Sleeping for
    "+str(sleep)+"s")
    print_percent_sleep(sleep)
  else:
    log("plain-test-"+str(int_hosts)+"-hosts.txt", "Sleeping for 20s")
    print_percent_sleep(20)

  ll = get_mem("adv-test-"+str(int_hosts)+"-"+str(int_dist)+"-hosts", ram_offset, ram_size)
  log("adv-test-"+str(int_hosts)+"-"+str(int_dist)+"-hosts.txt", ll)

def test_complex():
  ll = port_on(1)
  log("complex-test.txt", ll)

  hosts = lv_start_n(3)
  ll = "started: "+str(hosts)
  log("complex-test.txt", ll)

  log("complex-test.txt", "Sleeping for 4600s")
  print_percent_sleep(4600)

  hosts2 = lv_start_n(3)
  ll = "started: "+str(hosts2)
  log("complex-test.txt", ll)

  ll = lv_stop_host(hosts[1])
  log("complex-test.txt", ll)

  ll = lv_stop_host(hosts[2])
  log("complex-test.txt", ll)

  log("complex-test.txt", "Sleeping for 43200s")
  print_percent_sleep(43200)

  ll = get_mem("complex-test", ram_offset, ram_size)
  log("complex-test.txt", ll)

## main method of experiment
def main():
  # reset everything. Shutdown all vms, unpower the device.
  reset()

  for cnt in range(1,9):
    test_plain(cnt)
    reset()

  for cnt in range(1,9):
    for cnt2 in range(1,9):
      test_adv(cnt,cnt2)
      reset()

  if not debug:
    test_complex()
    reset()

main()
```

E Real Event Timelines Extracted from Logs

E.1 adv-test-1-4

Time	52:54:00:5C:AF:CA
1 462	up down
14879	memory dump

Table 4: Experiment Setting: One host is powered up and runs for approximately 460 seconds. Four hours later a memory dump is created.

E.2 adv-test-1-8

Time	52:54:00:5C:AF:CA
0 462	up down
29293	memory dump

Table 5: Experiment Setting: One host is powered up and runs for approximately 460 seconds. Eight hours later a memory dump is created.

E.3 plain-test-4

Time	52:54:00:40:f8:7B	52:54:00:5C:AF:CA	52:54:00:6F:B0:32	52:54:00:D5:4E:33
2	up	up	up	up
463	memory dump			

Table 6: Experiment Setting: Four hosts are booted. Approximately eight minutes later a memory dump is created.

E.4 adv-test-8-4

Time	52:54:00:00:AD:04	52:54:00:40:f8:7B	52:54:00:5C:AF:CA	52:54:00:6F:B0:32	52:54:00:74:68:4D	52:54:00:90:58:05	52:54:00:C1:02:83	52:54:00:D5:4E:33
5	up	up	up	up	up	up	up	up
469	down	down	down	down	down	down	down	down
14884	memory dump							

Table 7: Experiment Setting: Eight hosts are powered up and runs for approximately 460 seconds. Four hours later a memory dump is created.

E.5 adv-test-8-8

Time	52:54:00:00:AD:04	52:54:00:40:f8:7B	52:54:00:5C:AF:CA	52:54:00:6F:B0:32	52:54:00:74:68:4D	52:54:00:90:58:05	52:54:00:C1:02:83	52:54:00:D5:4E:33
5	up	up	up	up	up	up	up	up
468	down	down	down	down	down	down	down	down
29299	memory dump							

Table 8: Experiment Setting: Eight hosts are powered up and runs for approximately 460 seconds. Eight hours later a memory dump is created.

E.6 complex

Time	52:54:00:40:f8:7B	52:54:00:5C:AF:CA	52:54:00:6F:B0:32	52:54:00:74:68:4D	52:54:00:C1:02:83	52:54:00:D5:4E:33
1	up	up				up
4609			up	up	up	down
4610	down					
47855	memory dump					

Table 9: Experiment Setting: Three hosts are initially powered up. Approximately 1.25 hours later two of them are powered down and three new hosts are booted.

E.7 plain-test-8

Time	52:54:00:00:AD:04	52:54:00:40:f8:7B	52:54:00:5C:AF:CA	52:54:00:6F:B0:32	52:54:00:74:68:4D	52:54:00:90:58:05	52:54:00:C1:02:83	52:54:00:D5:4E:33
5	up	up	up	up	up	up	up	up
466	memory dump							

Table 10: Experiment Setting: Eight hosts are booted. Approximately eight minutes later a memory dump is created.

F Event Timelines as Extracted from Memory dumps

F.1 adv-test-1-4

Time	52:54:00:5C:AF:CA
45	DISCOVER
46	REQUEST
14883	memory dump

Table 11: Experiment Setting: One host is powered up and runs for approximately 460 seconds. Four hours later a memory dump is created.

F.2 adv-test-1-8

Time	52:54:00:5C:AF:CA
52	DISCOVER
53	REQUEST
29283	memory dump

Table 12: Experiment Setting: One host is powered up and runs for approximately 460 seconds. Eight hours later a memory dump is created.

F.3 plain-test-4

Time	52:54:00:D5:4E:33	52:54:00:5C:AF:CA	52:54:00:6F:B0:32	52:54:00:40:F8:7B
39			DISCOVER	
40			REQUEST	
43				DISCOVER
44	DISCOVER			
45	REQUEST	DISCOVER		REQUEST
46		REQUEST		
482	memory dump			

Table 13: Experiment Setting: Four hosts are booted. Approximately eight minutes later a memory dump is created.

F.4 adv-test-8-4

Time	52:54:00:5C:AF:CA	52:54:00:6F:B0:32
52	DISCOVER	
53	REQUEST	
63		DISCOVER
14883	memory dump	

Table 14: Experiment Setting: Eight hosts are powered up and runs for approximately 460 seconds. Four hours later a memory dump is created.

F.5 adv-test-8-8

Time	52:54:00:D5:4E:33	52:54:00:5C:AF:CA
51		DISCOVER
52		REQUEST
56	DISCOVER	
29282	memory dump	

Table 15: Experiment Setting: Eight hosts are powered up and runs for approximately 460 seconds. Eight hours later a memory dump is created.

F.6 complex

Time	52:54:00:6F:B0:32	52:54:00:40:F8:7B	52:54:00:C1:02:83	52:54:00:D5:4E:33	52:54:00:74:68:4D	52:54:00:5C:AF:CA
47		DISCOVER				
48				DISCOVER		
49		REQUEST				
38567	REQUEST					
38669			REQUEST			
38735						REQUEST
40143					REQUEST	
41528	REQUEST					
41743						REQUEST
42243			REQUEST			
43646					REQUEST	
44512	REQUEST					
45463			REQUEST			
45083						REQUEST
46785					REQUEST	
47439	REQUEST					
47882	memory dump					

Table 16: Experiment Setting: Three hosts are initially powered up. Approximately 1.25 hours later two of them are powered down and three new hosts are booted.

F.7 plain-test-8

Time	52:54:00:00:AD:04	52:54:00:6F:B0:32	52:54:00:40:F8:7B	52:54:00:90:58:05	52:54:00:C1:02:83	52:54:00:D5:4E:33	52:54:00:74:68:4D	52:54:00:5C:AF:CA
54						DISCOVER		
55						REQUEST		
63								DISCOVER
64					DISCOVER			
65			DISCOVER					REQUEST
66				DISCOVER				
67	DISCOVER		REQUEST	REQUEST	REQUEST			
68	REQUEST	DISCOVER						
69		REQUEST						
70							REQUEST	
483	memory dump							

Table 17: Experiment Setting: Eight hosts are booted. Approximately eight minutes later a memory dump is created.

G Utility for Lease Information Extraction

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import warnings
warnings.filterwarnings("ignore")

import sys
import os
import re

import time
import datetime
import subprocess

import svgwrite
from svgwrite import cm, mm, rgb, deg

def parse_date(str_date):
  re_non_space = re.compile(r'[^ ]+')
  lst_date = re_non_space.findall(str_date)

  dict_offsets = {'Jan':0, 'Feb':2678400, 'Mar':5097600, 'Apr':7776000, 'May':10368000,
  'Jun':13046400, 'Jul':15638400, 'Aug':18316800, 'Sep':20995200, 'Oct':23587200,
  'Nov':26265600, 'Dec':28857600}
  lst_hms = lst_date[2].split(':')

  return int(dict_offsets[lst_date[0]]) + 24 * 60 * 60 * (int(lst_date[1]) - 1) +
  int(lst_hms[0]) * 60 * 60 + int(lst_hms[1])* 60 + int(lst_hms[2])


def get_strings(str_file):
  proc = subprocess.Popen(["strings "+str_file], stdout=subprocess.PIPE, shell=True)
  (out, err) = proc.communicate()
  return out

def get_dates(str_mem):
  dict_ret = {}
  re_mem = re.compile(r'^([JFMASOND][aepuco][nbrylgptvc].*)\ \(none\)\ kern.notice\
  <25>DHCPS.*(DISCOVER\ from\
  ([0-9a-fA-F]{2}[:.-][0-9a-fA-F]{2}[:.-][0-9a-fA-F]{2}[:.-][0-9a-fA-F]{2}
  [:.-][0-9a-fA-F]{2}[:.-][0-9a-fA-F]{2})|REQUEST\ from\
  ([0-9a-fA-F]{2}[:.-][0-9a-fA-F]{2}[:.-][0-9a-fA-F]{2}
  [:.-][0-9a-fA-F]{2}[:.-][0-9a-fA-F]{2}[:.-][0-9a-fA-F]{2}))')
  for str_line in str_mem.split('\n'):
    if re_mem.match(str_line):
      lst_data = re_mem.findall(str_line)
      str_date = lst_data[0][0]
      date = parse_date(str_date)
      str_type = ""
      str_obj = ""
      if lst_data[0][3]:
        str_type = "REQUEST"
        str_obj = lst_data[0][3]
      else:
        str_type = "DISCOVER"
        str_obj = lst_data[0][2]

      dict_data = {"date":date, "type":str_type, "object":str_obj}
      if dict_ret.has_key(date):
        dict_ret[date].append(dict_data)
      else:
        dict_ret[date] = [dict_data]
  return dict_ret

def create_timeline(str_file):
  str_mem = get_strings(str_file)

```

```
  dict_date = get_dates(str_mem)

  lst_keys = dict_date.keys()
  lst_keys.sort()

  int_max_time = get_max_time(str_mem)

  create_svg(dict_date, str_file.split('/')[7],int_max_time)


def get_uniq_macs(dict_times):
  dict_macs = {}
  lst_macs = []
  counter = 0
  for key in dict_times.keys():
    for item in dict_times[key]:
      lst_macs.append(item['object'])
  lst_macs = list(set(lst_macs))
  lst_macs.sort()

  for mac in lst_macs:
    dict_macs[mac] = counter
    counter = counter + 1
  return dict_macs

def get_max_time(str_mem):
  max_time = 0
  re_mem = re.compile(r'^([JFMASOND][aepuco][nbrylgptvc].*)\ \(none\)\ kern.notice')
  for str_line in str_mem.split('\n'):
    if re_mem.match(str_line):
      lst_data = re_mem.findall(str_line)
      str_date = lst_data[0]
      date = parse_date(str_date)
      if max_time < date:
        max_time = date
  return max_time + 100

def scale_int(in_min, in_max, out_min, out_max, to_scl):
  range_in = float(in_max - in_min)
  range_out = float(out_max - out_min)

  factor = range_out / range_in
  return int( ((to_scl - in_min) * range_out) / range_in + out_min )


def create_svg(dict_times, str_image, int_max_time):
  dict_macs = get_uniq_macs(dict_times)

  print str_image

  dwg = svgwrite.Drawing(filename='/tmp/svg/'+str_image.split('.')[0]+'.svg',
  size=(1280,(10+len(dict_macs.keys()))*10))
  shapes = dwg.add(dwg.g(id='shapes', fill='red'))

  # add base labels
  for mac in dict_macs.keys():
    shapes.add(dwg.text(mac, insert=(10 , 10 * int(dict_macs[mac]) + 10 ), fill='black',
    class_='text',font_size=9 ))

  # add timeline
  shapes.add(dwg.rect(insert=(100,10 * len(dict_macs.keys())), size=(1000,1), fill='black'))
  shapes.add(dwg.text("Rel. Time t since device boot ( in seconds )", insert=(550, 80 + 10 *
  len(dict_macs.keys())), fill='black', class_='text',font_size=9 ))
  shapes.add(dwg.text("Black/Gray: Handed out leases, darkness determines certainty of
  presence Red: Requests Only", insert=(550, 90 + 10 * len(dict_macs.keys())), fill='black',
  class_='text',font_size=9 ))
  for i in range(0,1500,500):
    shapes.add(dwg.rect(insert=(100 + i ,10 * len(dict_macs.keys())), size=(2,10),
    fill='black'))
    int_tmp_counter = 0
```

```
    for j in str(scale_int(0, 1000, 0, int_max_time, i)):
      shapes.add(dwg.text(j, insert=(99 + i , 20 + 10 * len(dict_macs.keys()) + int_tmp_counter * 10), fill='black', class_='text',font_size=9 ))
      int_tmp_counter = int_tmp_counter + 1
  for i in range(0,1250,250):
    shapes.add(dwg.rect(insert=(100 + i ,10 * len(dict_macs.keys())), size=(1,8), fill='black'))
    int_tmp_counter = 0
    for j in str(scale_int(0, 1000, 0, int_max_time, i)):
      shapes.add(dwg.text(j, insert=(99 + i , 20 + 10 * len(dict_macs.keys()) + int_tmp_counter * 10), fill='black', class_='text',font_size=9 ))
      int_tmp_counter = int_tmp_counter + 1
  for i in range(0,1050,50):
    shapes.add(dwg.rect(insert=(100 + i ,10 * len(dict_macs.keys())), size=(1,4), fill='black'))
    int_tmp_counter = 0
    for j in str(scale_int(0, 1000, 0, int_max_time, i)):
      shapes.add(dwg.text(j, insert=(99 + i , 20 + 10 * len(dict_macs.keys()) + int_tmp_counter * 10), fill='black', class_='text',font_size=9 ))
      int_tmp_counter = int_tmp_counter + 1

  str_prt = "Time "
  for mac in dict_macs.keys():
    str_prt = str_prt + " & " + mac
  print str_prt + "\\\\"
  for timeslot in dict_times.keys():
    for item in dict_times[timeslot]:
      if item['type'] == "DISCOVER":
        offset = 100 + scale_int(0, int_max_time, 0, 1000, timeslot)
        hour = 4
        mac = item['object']
        if offset + hour > 1100:
          hour = hour - (offset + hour - 1100)
        shapes.add(dwg.rect(insert=(offset , 10 * int(dict_macs[mac])), size=(hour,9), fill='red'))
    for item in dict_times[timeslot]:
      if item['type'] == "REQUEST":
        offset = 100 + scale_int(0, int_max_time, 0, 1000, timeslot)
        hour = scale_int(0, int_max_time, 0, 1000, 3600)
        mac = item['object']
        if offset + hour > 1100:
          hour = hour - (offset + hour - 1100)
        shapes.add(dwg.rect(insert=(offset , 10 * int(dict_macs[mac])), size=(4,9), fill='black'))
        shapes.add(dwg.rect(insert=(offset , 10 * int(dict_macs[mac])), size=(hour,4), fill='black'))
        offset = offset + hour
        if offset + hour > 1100:
          hour = hour - (offset + hour - 1100)
        shapes.add(dwg.rect(insert=(offset , 10 * int(dict_macs[mac])), size=(hour,4), fill='grey'))
        offset = offset + hour
        if offset + hour > 1100:
          hour = hour - (offset + hour - 1100)
        shapes.add(dwg.rect(insert=(offset , 10 * int(dict_macs[mac])), size=(hour,4), fill='lightgrey'))

    str_prt = str(item['date']) + " & "
    #print timeslot
    #print dict_times[timeslot]
    for mac in dict_macs.keys():
      last = ""
      for item in dict_times[timeslot]:
        if item['object'] == mac and not last == item['object']+item['type']:
          last = item['object']+item['type']
          str_prt = str_prt + item['type']
      str_prt = str_prt + " & "
    print str_prt[0:-2] + "\\\\"
  print "\hline"
```

```
  print str(int_max_time)+" & \\multicolumn{"+str(len(dict_macs.keys()))+"}{c|}{memory dump}
  \\\\"
  dwg.save()

  print ""


if __name__ == "__main__":
  create_timeline(sys.argv[1])
```

H Example System Log Messages from Memory Dump

```
1 Jan  1 13:06:22 (none) kern.notice <25>DHCPC:GET ip:9164665a mask:fffffe0 gateway:91646641 dns1:9164600b dns2:91646016 static route:0 140
2 Jan  1 13:06:22 (none) kern.notice <25>Dynamic IP(DHCP Client) obtained an IP successfully  96
3 Jan  1 13:10:39 (none) kern.notice <25>DHCPS:Recv REQUEST from 52:54:00:6F:B0:32  86
4 Jan  1 13:10:39 (none) kern.notice <25>DHCPS:Send ACK to 192.168.1.103  76
5 Jan  1 13:11:21 (none) kern.notice <25>DHCPC Send REQUEST to server 9164602a with request ip 9164665a 107
6 Jan  1 13:11:22 (none) kern.notice <25>DHCPC Recv ACK from server 9164602a with ip 9164665a lease time 600 112
7 Jan  1 13:11:22 (none) kern.notice <25>DHCPC:GET ip:9164665a mask:fffffe0 gateway:91646641 dns1:9164600b dns2:91646016 static route:0 140
8 Jan  1 13:11:22 (none) kern.notice <25>Dynamic IP(DHCP Client) obtained an IP successfully  96
9 Jan  1 13:16:21 (none) kern.notice <25>DHCPC Send REQUEST to server 9164602a with request ip 9164665a 107
10 Jan  1 13:16:22 (none) kern.notice <25>DHCPC Recv ACK from server 9164602a with ip 9164665a lease time 600 112
11 Jan  1 13:16:22 (none) kern.notice <25>DHCPC:GET ip:9164665a mask:fffffe0 gateway:91646641 dns1:9164600b dns2:91646016 static route:0 140
12 Jan  1 13:16:22 (none) kern.notice <25>Dynamic IP(DHCP Client) obtained an IP successfully  96
13 Jan  1 10:41:22 (none) kern.notice <25>Dynamic IP(DHCP Client) obtained an IP successfully  96
14 Jan  1 10:42:47 (none) kern.notice <25>DHCPS:Recv REQUEST from 52:54:00:6F:B0:32  86
15 Jan  1 00:00:47 (none) kern.notice <25>DHCPS:Recv DISCOVER from 52:54:00:40:F8:7B  87
16 Jan  1 00:00:48 (none) kern.notice <25>DHCPS:Send OFFER with ip 192.168.1.105  83
17 Jan  1 00:00:48 (none) kern.notice <25>DHCPS:Recv DISCOVER from 52:54:00:D5:4E:33  87
18 Jan  1 00:00:49 (none) kern.notice <25>DHCPS:Send OFFER with ip 192.168.1.101  83
19 Jan  1 00:00:49 (none) kern.notice <25>DHCPS:Recv REQUEST from 52:54:00:40:F8:7B  86
20 Jan  1 00:00:49 (none) kern.notice <25>DHCPS:Send ACK to 192.168.1.105  76
21 Jan  1 00:00:49 (none) kern.notice Jan  1 00:00:49
22 Jan  1 12:01:22 (none) kern.notice <25>Dynamic IP(DHCP Client) obtained an IP successfully  96
23 Jan  1 12:06:21 (none) kern.notice <25>DHCPC Send REQUEST to server 9164602a with request ip 9164665a 107
24 Jan  1 12:06:22 (none) kern.notice <25>DHCPC Recv ACK from server 9164602a with ip 9164665a lease time 600 112
25 Jan  1 12:06:22 (none) kern.notice <25>DHCPC:GET ip:9164665a mask:fffffe0 gateway:91646641 dns1:9164600b dns2:91646016 static route:0 140
26 Jan  1 12:06:22 (none) kern.notice <25>Dynamic IP(DHCP Client) obtained an IP successfully  96
27 Jan  1 12:07:26 (none) kern.notice <25>DHCPS:Recv REQUEST from 52:54:00:74:68:4D  86
28 Jan  1 12:07:26 (none) kern.notice <25>DHCPS:Send ACK to 192.168.1.104  76
29 Jan  1 12:11:21 (none) kern.notice <25>DHCPC Send REQUEST to server 9164602a with request ip 9164665a 107
30 Jan  1 12:11:22 (none) kern.notice <25>DHCPC Recv ACK from server 9164602a with ip 9164665a lease time 600 112
31 Jan  1 12:11:22 (none) kern.notice <25>DHCPC:GET ip:9164665a mask:fffffe0 gateway:91646641 dns1:9164600b dns2:91646016 static route:0 140
32 Jan  1 12:11:22 (none) kern.notice <25>Dynamic IP(DHCP Client) obtained an IP successfully  96
33 Jan  1 12:16:21 (none) kern.notice <25>DHCPC Send REQUEST to server 9164602a with request ip 9164665a 107
34 Jan  1 12:16:22 (none) kern.notice <25>DHCPC Recv ACK from server 9164602a with ip 9164665a lease time 600 112
35 Jan  1 12:16:22 (none) kern.notice <25>DHCPC:GET ip:9164665a mask:fffffe0 gateway:91646641 dns1:9164600b dns2:91646016 static route:0 140
36 Jan  1 12:16:22 (none) kern.notice <25>Dynamic IP(DHCP Client) obtained an IP successfully  96
37 Jan  1 12:21:21 (none) kern.notice <25>DHCPC Send REQUEST to server 9164602a with request ip 9164665a 107
38 Jan  1 12:21:22 (none) kern.notice <25>DHCPC Recv ACK from server 9164602a with ip 9164665a lease time 600 112
39 Jan  1 12:21:22 (none) kern.notice <25>DHCPC:GET ip:9164665a mask:fffffe0 gateway:91646641 dns1:9164600b dns2:91646016 static route:0 140
40 Jan  1 12:21:22 (none) kern.notice <25>Dynamic IP(DHCP Client) obtained an IP successfully  96
41 Jan  1 12:21:52 (none) kern.notice <25>DHCPS:Recv REQUEST from 52:54:00:6F:B0:32  86
42 Jan  1 12:21:52 (none) kern.notice <25>DHCPS:Send ACK to 192.168.1.103  76
43 Jan  1 12:26:21 (none) kern.notice <25>DHCPC Send REQUEST to server 9164602a with request ip 9164665a 107
44 Jan  1 12:26:22 (none) kern.notice <25>DHCPC Recv ACK from server 9164602a with ip 9164665a lease time 600 112
```

I Possible DHCPD in-memory Lease file

```
01d4f4c0  00 00 00 01 52 54 00 90  58 05 00 00 00 00 00 00  |....RT..X.......|
01d4f540  00 00 00 01 52 54 00 00  ad 04 00 00 00 00 00 00  |....RT..........|
01d4f5c0  00 00 00 01 52 54 00 c1  02 83 00 00 00 00 00 00  |....RT..........|
01d4f640  00 00 00 01 52 54 00 74  68 4d 00 00 00 00 00 00  |....RT.thM......|
01d4f6c0  00 00 00 01 52 54 00 40  f8 7b 00 00 00 00 00 00  |....RT.@.{......|
01d4f740  00 00 00 01 52 54 00 d5  4e 33 00 00 00 00 00 00  |....RT..N3......|
01d4f940  00 00 00 01 52 54 00 5c  af ca 00 00 00 00 00 00  |....RT.\........|
01d4f9c0  00 00 00 01 52 54 00 6f  b0 32 00 00 00 00 00 00  |....RT.o.2......|
01e05af0  81 d9 52 54 00 00 00 00  00 00 00 00 81 e0 58 94  |..RT..........X.|
01edbf80  5f 50 4f 52 54 00 00 00  00 00 00 00 00 00 00 00  |_PORT...........|
01f50f90  00 00 00 06 52 54 00 90  58 05 00 00 00 00 00 08  |....RT..X.......|
01f50fd0  00 00 00 00 00 00 00 00  52 54 00 90 58 05 00 00  |........RT..X...|
01f58b30  50 4f 52 54 00 00 00 00  32 32 37 20 00 00 00 00  |PORT....227 ....|
01f58b40  45 50 52 54 00 00 00 00  32 32 39 20 00 00 00 00  |EPRT....229 ....|
01f5c8e0  81 e0 52 54 00 00 00 00  00 00 00 00 81 15 59 00  |..RT..........Y.|
01f66430  00 00 00 00 52 54 00 90  58 05 00 00 00 00 00 00  |....RT..X.......|
01f66910  00 00 00 00 52 54 00 90  58 05 00 00 00 00 00 00  |....RT..X.......|
01f691b0  00 00 00 00 52 54 00 5c  af ca 00 00 00 00 00 00  |....RT.\........|
01f691f0  52 54 00 6f b0 32 00 00  00 00 00 00 00 00 00 00  |RT.o.2..........|
01f69260  00 00 00 00 00 00 00 00  52 54 00 40 f8 7b 00 00  |........RT.@.{..|
01f692a0  00 00 00 00 52 54 00 74  68 4d 00 00 00 00 00 00  |....RT.thM......|
01f692e0  52 54 00 c1 02 83 00 00  00 00 00 00 00 00 00 00  |RT..............|
01f69350  00 00 00 00 00 00 00 00  52 54 00 90 58 05 00 00  |........RT..X...|
```