Shepherding plan for “Regulating ARM TrustZone Devices in Restricted Spaces”

In the plan below, feedback that we received from the PC appear in red font, our own responses/comments to this feedback (where applicable) appear in blue font, and our action items appear in green font.

Feedback explicitly called out in the PC meeting summary appears first, since our perception was that these were the most important ones to be addressed in our revision.

Task 1 (PC discussion comments, Review-37A, Review-37B, Review-37C, and Review-37D)

"Add discussion/numbers on the frequency at which the host can request the client for the verification token."

We did not include a discussion on the frequency with which the host can request the verification token because this necessarily involves a tradeoff. The more frequent the requests, the greater its confidence in the guest device's compliance.

We will explicitly mention this tradeoff, but are reluctant to specify what the frequency should be, since this depends on the host. The host can request it very often, and obtain stronger guarantees on the guest, or request it less often at the cost of having a larger window of vulnerability. We will quantify the size of the verification token and the time taken to compute it. This will give an idea of the overheads on the guest device to transmit it each time. Tying in more broadly to a point that reviewer-37A and reviewer-37B noted, about time-of-check to time-of-use, TOCTTOU is indeed one of the known shortcomings of trusted computing technology (shared also by our work) and we will explicitly include a discussion about this.

Addressed with additional text in Section 2.2 and Section 6, and new experimental data in Fig 6.

Task 2 (PC discussion comments and Reviewer-37B)

"the authors are asked to provide a more detailed discussion on how the host can guarantee that the client kernel image is rootkit-free" and "the current writing of the paper is vague."

Thanks in particular to reviewer-37B for pointing out that our description was vague. Indeed, our intention was to convey the breadth and power of remote memory operations (in Section 3). It was not to convey that we've done this all in our prototype. What we have done in our prototype is specified in Section 6.

We will clearly distinguish in Section 3 what we have done in our prototype, and what we are proposing as other applications of remote memory operations. As suggested by reviewer-37B, we will try to remove all instances of uncertain connectives such as "can" and "could" and try to be more concrete in our discussions in Section 3.

Addressed with many minor changes throughout the paper (not marked up), as well as localized changes in Section 3 of the paper.

Task 3 (PC discussion comments and Reviewer-37B)

"what guarantees does the current implementation of the vetting service provide?"

Section 5's description is exactly what we implemented in our vetting service. Since it looks like this wasn't clear from our writing, we will explicitly state this, and include experimental data about the vetting service (also requested by reviewer 37-B) In particular, reviewer-37B is right that the vetting service is in the TCB, but we did not quantify it in Figure 5 because we were only interested in the TCB components on the guest.

We will clearly specify that the vetting service is part of the TCB, but that because it isn't part of the guest, we didn't aim to minimize its size (as part of our design goals). We will include the size of the vetting service (either in Section 5 or in Section 6). We cannot include LOC count, since Hex-Rays (on which we built it) is closed-source. Nevertheless, we will include a suitable metric. We will also provide some performance numbers for our read-safety and write-safety policies (in Section 5).

Addressed with new text in Section 5 and 6, and some performance numbers included in Fig 6.

Task 4 (Reviewer 37A)

"Consider the following strawman design as an alternative. The OS has a safe mode in which it can boot. When booted in the safe mode, the OS performs a number of modifications such as disabling drivers (or installing dummy ones), locking certain memory pages, etc... At check-in, SW forces an OS shutdown followed by a boot up in safe mode. Can the paper compare their system with this strawman?"

and

"I think nullifying interfaces and dummy drives are very brittle approaches to securing devices. Devices are not built to be robust to such "tricks". Some of the results in Figure 7 (this should be called a table, no?) confirm my suspicions. Why not assign the devices to SW and have SW refuse service? Or why not reboot the OS and shut off these devices?"

Reassigning peripherals to the SW is indeed a neat solution that was also brought to our attention earlier by one of our collaborators. It is possible via various methods, depending on the specific device. One popular method is by leveraging the APB-to-AXB bridge. However, the peripheral reassignment does not persist across reboots, and it is unclear how to provide “proofs of compliance” (the way we do using verification tokens) to the host using this method. Moreover, for rootkit detection, one still needs to perform remote memory access, and our paper argues that the same remote memory access interface can be repurposed for peripheral control.

We will add a discussion of peripheral reassignment, and the tradeoffs of using that method vs our method (e.g., dummy drivers) in the “Related Work and Alternatives” section.

Addressed in Section 7.

Task 5 (Reviewer 37A)

I would suggest the paper use established terms from trusted computing rather than coining their own terms. For example, the term "software attestation" is well-known and I think that's what the paper means when using the term "verification token". Also, the process by which data structures are

inspected is typically referred to as "measurement".

While software attestation has been used to attest to the code stack, we're not aware of prior work on using this to attest dynamic data structures in memory (aside from some work on property-based attestation that has done this). We will definitely include a discussion in the paper drawing parallels between our work and software attestation, e.g., using TPMs.

Addressed in Section 2.2

Task 6 (Reviewer 37A)

K_{Dev} is not a feature of ARM TrustZone. Gadget 2008 is an imaginary device (see page 6-1). ARM TrustZone does not require provisioning each device with a secret key. Most devices out there do not have this key. Please read the ARM TZ spec carefully.

It actually wasn't too clear to us from the TZ spec whether this was a proposed part of the specification, or an option on ARM TZ devices. Our own board did not have such a K_{Dev}, and so our implementation of REM Suspend does everything except the encryption using K_{Dev} (the last step). Thank you for pointing this out to us.

In the revision, we will make clear that REM Suspend works on devices that provide such a K_{Dev}, which is referenced in the ARM TZ spec as an example use case in Gadget 2008.

Addressed in Section 4.4 and Section 6.

Task 7 (Reviewer 37A)

"What happens in the scenario after the check-in when the device discovers malware? Could the paper explain how the device is quarantined, and how it is cleaned?"

We do not specify a particular policy for this in the paper, since different hosts can have different policies for this. We do not wish to include a discussion for how the device is cleaned, since we think that is an orthogonal topic, and much too challenging to be considered within the confines of this paper.

In Section 2, we will include a brief discussion on the various possibilities for the host to "quarantine" the device.

Addressed in Section 2.2.

Task 8 (Reviewer 37A)

"Who acts as the certifying authority? Consider NSA/FBI/government as being the host, and visitors (e.g., University professors) being the guest. Could the paper propose several examples on who should act as the certifying authority?"

and

"What prevents a MITM for the algorithm in Figure 4? Why isn't it possible for malware to fake a

certificate for PubKeyG? Is there a database of public keys of all devices out there? Who creates certificates?"

We will add some comments in Section 4.1 (Authentication) on who could act as CAs. We believe that this discussion will address both of the concerns above. With certificates issued by a CA that is trusted by both the host and the guest, MITM attacks are no longer possible.

Addressed in Section 4.1

Task 9 (Reviewer 37A)

"Important related work to this paper:

- "AdAttester". Mobisys 2015.
- "Splitting the bill". HotMobile 2013.
- "Mobile Trusted Computing". IEEE 102(8): 1189-1206 (2014).
- "fTPM". MSR Tech Report. "

We will cite these in related work.

Addressed in Section 7, as well as in Section 4.4 (citations to concepts from fTPM)

Task 10 and Comments on Reviewer 37B's feedback

"In page 4, they say that host must check the memory snapshot to make sure /dev/kmem is not exposed or modules are not allowed. Or they say that at check-in, the host can check that an anti-malware is running tha t can detect malicious behavior. Unfortunately, these are not effective. A simple reboot of the guest is enough to revert all of these."

The fact that reboots can undo these changes is exactly why our paper introduces the REM-suspend mechanism, which we believe already addresses this comment from the reviewer.

"Moreover, even without a reboot, there are many other ways for a malicious guest (which has complete control of the guest normal world) to fool the host and revert the changes to the memory. For example, an additional ioctl command can be added to one of the drivers that can be used to redo the changes. This additional ioctl will not require any changes to the driver callback list and cannot be easily detected by the host. As for the anti-malware, it can be turned off after the check. "

The task of inspecting the kernel is to make sure that such malicious entities (e.g., a driver with the additional ioctl) are not in kernel memory. We're not clear how the ioctl in the driver can avoid a footprint in the driver callback list. If the ioctl has to be invoked, it has to be registered by the driver via the callback list. If the driver is unloaded (e.g., because we install a new dummy driver), so will the callback.

"There is also a major problem with the practicality of nullifying or redirecting device driver callbacks. How about aliases? A malicious guest can have many aliases (i.e., function pointers) pointing to the same driver callbacks. It can even use temporary aliases. That is, it can store the physical address of the callback and map it to some virtual address just before calling it and unmapping it afterwards. Does the host currently check to find all of these possible attack vectors. Alias analysis in the kernel can be quite

compute extensive and might require a significant portion of the kernel memory to be transferred to the host. This will make the solution even less practical. ”

A detailed forensic analysis of the normal world's snapshot can establish that the normal world satisfies certain invariants. Such forensic analyses have been described in prior work (e.g., Baliga, Ganapathy and Iftode), and we reference it in the paper. We did not implement them for this paper because we just wanted to show a reliable way to acquire a snapshot of the device's memory (and we assumed that prior work on snapshot analysis would be directly applicable). For our prototype, we implemented a simple snapshot analysis focusing on the system call table alone (described in our Section 6). We do transfer all of kernel memory to the host and quantify the time overhead to do so in Section 6.

We will try to include text in the paper so that all the clarifications that we made above appear at suitable locations in the paper.

Addressed in Section 3

Task 11 (Reviewer 37B)

“Another major shortcoming of this approach is its inability to deal with mmaped memory. Mmap can be used to map part of a peripheral device register space to user space allowing part of the device driver to be pushed to user space if needed. Since the proposed solution only analyzes the kernel memory, it will not be able to detect such mmaped regions into the user space. This opens an unchecked attack surface for malicious code in the guest. ”

Thank you for raising this point. We will include a discussion of mmap. We suspect that even if device memory is mmaped, without a driver in place, applications will be unable to use the peripheral properly. But this is not something that we have actually experimented with to date. We will include a discussion for this point.

Addressed together with Task 10 in Section 3

Task 12 (Reviewer 37B)

“The guest needs to transfer the kernel memory to the host. Authors argue that the user memory is not transferred to the host for privacy reasons. However, the content of kernel memory contains sensitive and private information as well. For example, the buffer cache, which is a cache for disk content, is in the kernel memory. Also, the buffers in the networking stack that hold the t ransmitted/received packets are in the kernel memory and might not be zeroed out. There is also a lot of info on the applications running in the system. Unfortunately, the proposed solution does not offer any insights on how protect such sensitive information from a malicious host. ”

We do not have defenses in our prototype vetting service for the buffer cache or buffers in the networking stack, but believe that more sophisticated vetting algorithms can prevent this information from going to the host. For example, to vet safety, the guest need only provide the pages containing code, control data, and certain non-control data (e.g., pointer data). Buffers storing application data are not critical to establish normal world kernel security. The host could only request pages that do not contain these buffers, and the vetting service could make sure that the host's requests do not request buffers storing user data.

We will include a brief discussion of more complex vetting policies, but make it clear that we have not implemented them in our prototype.

Addressed in Section 5

Task 13 (Reviewer 37B)

"Does the proposed solution work if the guest adopts Address Space Layout Randomization (ASLR) (as new Android devices do)? Will the host be able to still find the device driver callbacks by analyzing the kernel memory snapshot? "

With sufficient information about the Android kernel version, yes, the host can do so. This is because recovering data structure layout happens by recursively following pointers in data structures and not by directly analyzing page layouts. The data structure recovery algorithm discussed in prior memory snapshot analysis algorithms works irrespective of whether ASLR was deployed or not.

We will include in Section 3 a sentence or two about ASLR and that prior techniques can recover data structures in the presence of ASLR.

Addressed in Section 3.

Task 14 (Reviewer 37D and Reviewer 37E)

"I would have liked to see a short survey of the types of configurations or desirable policies that are common for restricted spaces to help motivate the design of the system. For example, if most of such policies entail shutting down the network interfaces (WiFi, cellular) and peripherals such as camera, microphone, USB (as shown in Figure 7), then such design can be more customized to support these common configurations to reduce unnecessary overhead. "

and

"For the general MobiSys community, it may be helpful for the authors to better highlight the design challenges. Also, it would be helpful to discuss the different types of restricted spaces, how the policies may differ among hosts and guest devices, and more importantly, how such differences may impact or limit the proposed solution. "

We did have such a discussion of policies in prior drafts of the paper, with different scenarios and showing how host control over peripherals could help with those scenarios. However multiple readers complained that it distracted from the general flow of the paper, and so we removed it.

Space permitting, and if it does not distract from the general flow of the paper, we will include a brief discussion of examples of policies. However, since we have had negative feedback with examples of policies included in the paper, we are not guaranteeing that we will include this in the camera-ready draft of the paper.

Not addressed. We chose not to address this because we do not see a logical place for this discussion in the paper. There is already some discussion of what peripherals can be disabled under various settings (e.g., in Section 2.2, point (3)), and we feel that adding this as a separate discussion would detract from the flow of the paper, in addition to contributing further

discussion that is not directly relevant to the concepts developed in the paper. We have tried adding some text in the past to this effect, but without success. Please note that we did not guarantee to make this change anyway.

Task 15 (These are our own, not in response to any specific comments)

- There are some recently-proposed cache-only attacks that our current architecture, which is based on memory introspection, cannot defend against ("CacheKit" Euro S&P'16 paper). We may include a brief discussion of such attacks.

- We plan to include a discussion of physical memory vs virtual memory-based introspection in Section 4.2, and their relation to ATRA attacks (ACM CCS'14).

Added discussion in Section 3 and Section 4.2

Comments that we do not intend to act upon

- Reviewer 37C: "The read/write mechanism seems troublesome to me. Wouldn't disabling devices through writing memory lead to system instability?"

  and

  Reviewer 37D: "The overhead of the system is not studied from the perspective of the disturbance on the allowed apps running on the device."

  We do not intend to act on this since we believe Figure 7 in the paper already answers this question.

- Reviewer 37A: "Why not use send software attestations from the device to the host and have the host analyze them rather than doing all this expensive work on the device?"

  We are not sure what the reviewer means here. Our design already sends software attestations from the device to the host, and requires the host to analyze them. We do not do any expensive work on the device. Perhaps this wasn't clear to the reviewer.