Prove that LSM-based provenance capture is guaranteed to detect a security breach.

Aarti Kashyap
19115724

1 Overview of the Proposal

CamFlow is highly configurable Provence capture system. CamQuery modifies CamFLow with a novel query module mechanism that enables runtime provenance analysis and even mediation of system events. The design and implementation of CamQuery extend the guarantees of past provenance monitors to support runtime provenance analysis in terms of completeness and accuracy. There has been prior work in the formal verification of LSM placement ensuring that all interactions are being captured between kernel objects. However, my intention is to use the methodology used by Georget et al. to prove the reliability of OS-level information control flow structure systems in Linux to prove if it also captures the security properties of the LSM modules.

1.1 Current status

The main objective of completeness in this case should guarantee that all flows between kernel objects are properly recorded. The LSM framework was originally implemented to support Mandatory access control schemes, however, not information flow tracking. Recent work by Georget et al. [4] demonstrated, through static analysis of the kernel code base, that the LSM framework is applicable to information flow tracking, and that by adding a small number of LSM hooks, it was possible to properly intercept all information flows between kernel objects.
Thomas et al. maintain a patch based on the work by Georget et al. maintain a patch [1] to the LSM framework that allows CamFlow, and by extension CamQuery, to provide stronger guarantees than do previous whole-system provenance capture mechanisms.

1.2 Research Questions

1.2.1 RQ1

Based on the methodology proposed by Georget et al. I intend to explore if the current LSM interface captures all security-related flows in the kernel. The authors of CamQuery have omitted a complete security analysis of the system and refer to previous work in the literature. Hence, following the static analysis approach and experimentation, Georget at al. proposed a revision of LSM with some hooks added, and some existing hooks relocated. Does this approach ensure that the LSM interface is capturing all security-related flows in the kernel?

1.2.2 RQ2

Based on RQ1, if the model captures all security-related flows then it must show up as an anomaly in the provenance graph.

1.3 Additional Thoughts

The reason these are important research questions is because Georget et al. approach covers the notion that their formal model either covers all paths and if does not cover all paths, it's impossible. Since the patch applied by Thomas et al. now ensures that all execution paths data is being collected, intuitively it should also catch all security-related flows in the LSM interface. However, it is worth noting if there are some unexpected outcomes. Similarly for RQ2, if it is capturing all the security-related flows, then again intuitively it must show up as an anomaly in the graph.

2 Related Work

I have to delve a little deeper into the related works section, as I feel it's not enough yet.

2.1 Background Reading

CamFlow is a Linux Security Module (LSM) designed to capture data provenance for the purpose of system audit. The provenance capture mechanism is highly configurable, and can fit many applications' particular needs. CamFlow can stack with existing security modules such as SELinux. Another facility provided by CamFLow is the ability to fine-tune the provenance information it captures. Continuing the work on CamFlow, CamQuery was introduced, whose goal is to bridge the gap between run-time security monitoring and post-hoc forensic analysis. A very small part of the paper, focuses on the completeness and the accuracy properties of the design and implementation of CamQuery and this is where I come in.

2.2 Contextual work

CamFlow [6] is a Linux Security Module (LSM) designed to capture data provenance for the purpose of system audit. The provenance capture mechanism is highly configurable, and can fit many applications' particular needs. CamFlow can stack with existing security modules such as SELinux. CamQuery [7], is a framework that supports runtime analysis of provenance and thus enables its practical use for a variety of security applications. CamQuery pairs a runtime kernel-layer reference monitor – expanding and modifying CamFlow with a novel query module mechanism that enables runtime provenance analysis and even mediation of system events. The proofs provided by Georget et al. also seem to be valid for a previous version of LSMs. However, they are incorporated into the current framework assuming they work. So the first thing we want to check is, if the proofs work with the newer version of LSMs. Also, since they are securing the reliability properties of the system, I should be checking for the security properties too.

2.3 List of papers I intend to read

The following are the paper that I have read [4] [7] and the papers which I intend to read [6] [2] [3] [5] in order to get a deeper understanding of the system.

3 Conclusion Format

The results can be in one of the following categories

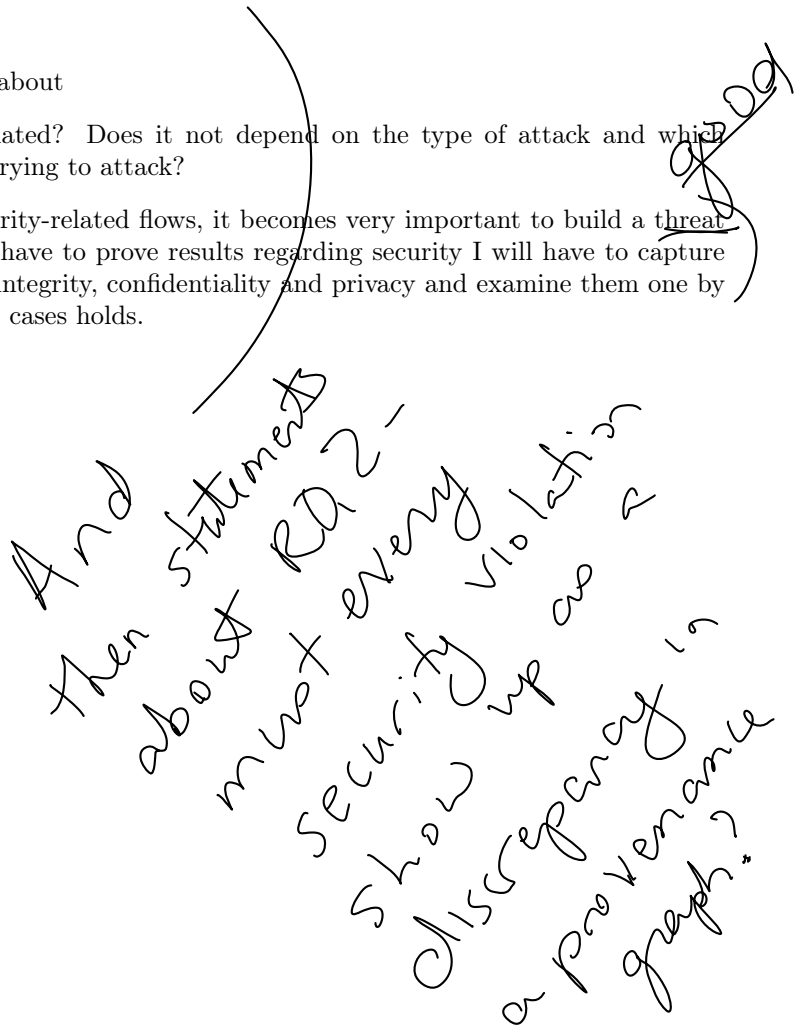
1. If the formal model is correct
   (a) The same methodology should capture all the security-related flows ideally, since it claims to capture all the execution paths and the ones which cannot be captured are impossible.
   (b) Despite the formal model being correct it is not capturing the security-related flow, which means it's going in one of the impossible paths and there is a vulnerability somewhere.
2. There is some flaw with the formal model
   (a) Despite the flaw, we are able to capture all security-related flows.
   (b) Since, there is a flaw with the formal model, there are some paths it's not executing and a few of those are security-related.

Logically speaking these are the only four cases possible. We just need to be confirm with proof-of-concept which one it is.

3.1 Additional Thoughts

Some other logical points to think about

1. Are not all flows security-related? Does it not depend on the type of attack and which vulnerability the attacker is trying to attack?
2. When we try to look for security-related flows, it becomes very important to build a threat model for the system, so if I have to prove results regarding security I will have to capture the three main components: integrity, confidentiality and privacy and examine them one by one to see which of the above cases holds.



4 Experimental Setup

4.1 Which experiments will I conduct? Why? What question is each answering?

1. In order to familiarize myself with the system, I will run the patch and observe the data collected.

2. I will observe the data collected and see if there is some anomaly I can catch manually. By anomaly I mean if there is some case that I can based on observation see is missing.
   Thomas et al. in CamQuery manually prove the accuracy properties of the systems, hence, I think it is important to understand the system well before running any random tests and the data it produces.

3. The next step for me is to fix the security models. The reason is that I will start from a one model and then I can extend it to different models as I test the system. It will help me also capture if all security-flows are violated or only a specific ones.

4. Based on the security model, I will manually find the execution paths which need to be monitored.

5. After manually finding the paths, using the methodology provided by the Georget et al. I will run the patches to see if those flows are being recorded.

6. The next steps depend on the results I obtain.

4.2 Conditional experiments

1. If there are some security-related flows which are not being followed, I will try to categorize them and try to find some pattern, to confirm if all the category of those attacks are the ones which are being not captured by the methodology or if its just one.
   Understanding why that path is not being covered is important in order to find the flaw with the methodology.

2. However, if for my specific security model all flows are being captured, I will move ahead with different sets of security models based on Bates et al. [2] security analysis work and see if I have covered all aspects.

3. If I manage to cover all the cases, and can see that the security flows are being captured I will move to part 2, where I will try to see if the graph shows the anomaly in case of a security violation.

4. For the above point I will conduct other sets of experiments, however, before that I am interested in the part one of this version.

4.3 How will I confirm if my measurements are correct?

1. After every set of data collection by running the tool, I will re run the tool just in case.

2. I will try to cover as many cases as possible for data collection in the initial stages in order to find some sort of pattern between the data.

4.4 Main questions I need answered before I start the "research" work

1. How does the tool work? (Not only in theory but in practice), since after KLEE, I am a little nervous about trying out this tool.

2. I want to find the control flow based on the tests I conduct, since, the approach by George et al. was opposite, based on the control flow he added and relocated the hooks.
   I want to collect data and try to understand the control flow based on the data I obtain. Why? I think this will help me get a better understanding of the system. I can then manually map my obtained control flow with the real control flow. This way I can also check if the data I am collecting is accurate.

3. Does the tool work for the current versions of LSMs as predicted by the Georget et al?

After these sets of tests, I can now safely assume that I am familiar with the system and know how the tool works and can proceed to answer the next set of questions.

4.5 Main Questions I need answered after I am familiar with system

1. This goes back to the RQs. However, I intend to do fowrward and backward chaining combined.
   While conducting tests, I will also study the mathematical model proposed by Georget et al. in order to find possible cases which have been left out during the proposal of completeness and accuracy lemmas. The reason is completeness and accuracy does not necessarily guarantee correctness. Hence, finding those subtle side cases by understanding the proofs and abstractions will be a good exercise.

2. I want to categorize every type of property check those properties one at a time, two or/and three at time. Is it possible to hold all without violating any one, since reliability and security do not always go hand in hand. Right? It can but it might not.

5 Resources needed

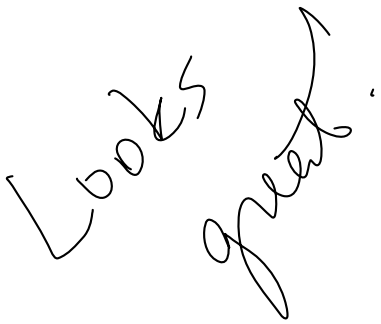
The resources required for the research are:

1. CamFlow patch
2. My Linux system

There are no additional resources I will require for the initial part. However, if I do happen to find an anomaly in the data, then I will have to look into proofs and might require additional help there.

6 Schedule

1. Before 1st Status Meet (Feb 5th - Feb 11th) - Be familiar with the tool and how it works and run atleast a two tests to observe the data for analysis.
2. Before 2nd Status Meet (Feb 15th - March 4th) - Try to verify the security related flows. Come up with an answer to RQ1, in order to understand where to proceed next.
3. Before In class presentation (March 5th - March 26th) - Wrap up RQ1 no matter what set of results are obtained after the experiments and present the in depth analysis of captured security-related flows in the lectures.
4. First draft due (March 29th) - Present results and analysis of RQ1 and the exact step by step approach for analysis of RQ2. Since by now I will have the results from Stage 1, I can plan how I want to proceed for Stage 2 to understand the relation between the graph and the security violation.
5. Final Submission 23rd April - Wrap results obtained from both RQs and conclude my work.



References

[1] Camflow information flow patch. In *https://github. com/CamFlow/information- flow- patch.*

[2] Adam Bates, Dave Tian, Kevin R. B. Butler, and Thomas Moyer. Trustworthy whole-system provenance for the linux kernel. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, pages 319–334, Berkeley, CA, USA, 2015. USENIX Association.

[3] Laurent Georget, Mathieu Jaume, Guillaume Piolle, Frédéric Tronel, and Valérie Viet Triem Tong. Information flow tracking for linux handling concurrent system calls and shared memory. In Alessandro Cimatti and Marjan Sirjani, editors, *Software Engineering and Formal Methods*, pages 1–16, Cham, 2017. Springer International Publishing.

[4] Laurent Georget, Mathieu Jaume, Guillaume Piolle, Frédéric Tronel, and Valérie Viet Triem Tong. Verifying the reliability of operating system-level information flow control systems in linux. In *Proceedings of the 5th International FME Workshop on Formal Methods in Software Engineering*, FormaliSE '17, pages 10–16, Piscataway, NJ, USA, 2017. IEEE Press.

[5] Xueyuan Han, Thomas Pasquier, and Margo Seltzer. Provenance-based intrusion detection: Opportunities and challenges. In *Workshop on the Theory and Practice of Provenance (TaPP'18)*. USENIX, USENIX, 2018.

[6] Thomas Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, David Eyers, Margo Seltzer, and Jean Bacon. Practical whole-system provenance capture. In *Symposium on Cloud Computing (SoCC'17)*. ACM, ACM, 2017.

[7] Thomas Pasquier, Xueyuan Han, Thomas Moyer, Adam Bates, Olivier Hermant, David Eyers, Jean Bacon, and Margo Seltzer. Runtime analysis of whole-system provenance. In *Conference on Computer and Communications Security (CCS'18)*. ACM, 2018.