Maintaining Authorization Hook Placements Across Program Versions

Nirupama Talele*, Divya Muthukumaran†, Frank Capobianco* Trent Jaeger*, Gang Tan*

*Penn State University
Email: {nrt123,fcc110,tjaeger,gtan}@cse.psu.edu

†Imperial College, London
Email: divya@doc.ic.ac.uk

We examine the problem of maintaining security code across program versions. There are now several cases where programmers manually retrofit their programs with security code, such as authorization mechanisms. However, programs evolve, so a challenge for programmers is to determine whether their security code remains correct across multiple versions of the program. The insight of this work is that programmers can use the constraints on the authorization policies that can be enforced in one version of the program to limit their effort in validating authorization hook placements in later versions. We develop a tool we call HEIMDAL to implement this insight, finding that a modest number of authorization constraints require review across several versions of the X window server program.

For various programs like servers that handle user data [1], [2], [6], [9], authorization mechanisms ensure that an access control policy is enforced on requests submitted by program clients. Currently, programmers add authorization hooks to their programs manually, which gives rise to two challenges. First, programmers must validate that the authorization hooks thus placed provide complete mediation over security-sensitive operations that is sufficient to enforce program's authorization policies. Second, programmers must ensure that the above validation holds as the program evolves across versions. Researchers have proposed methods to detect errors in existing hook placements that leverage knowledge of program's security-sensitive operations [7], [8], [10]. However, most of these methods require programmers to provide low-level program information, which they are often unwilling to do.

In this paper, we explore how to maintain authorization hook placements across versions. We propose that an authorization hook placement process should produce a set of constraints on the policies that can be enforced by the program, called *authorization constraints* [4]. An authorization constraint relates a pair of security-sensitive operations to a constraint on the sets of subjects authorized to perform them both. We have identified authorization constraints for subsumption or equivalence. First, if a security-sensitive operation o_1 is allowed for set of subjects which is a subset of subjects that are allowed for operation o_2, then we say that o_2 *subsumes* o_1 with respect to the policy. Second, if the subjects that are allowed to perform o_1 are the same as the subjects allowed for o_2 then we say that the two operations are *equivalent* with respect to the policy. The precise definitions can be found in [4].

We present HEIMDAL, a static analysis tool for validating authorization hook placements or identifying parts of the program that fail compliance and further leverage it to validate the correctness of the placement across versions. HEIMDAL first computes a *candidate hook placement* for the program sufficient to achieve complete mediation for its security-sensitive operations [3]. The programmers then apply HEIMDAL to produce the authorization constraints for the program semi-automatically and generate a minimal authorization hook placement for the program sufficient to enforce any access control policies that comply with the authorization constraints [4]. The intuition is that these authorization constraints can be leveraged to focus programmer effort in validating placements for subsequent versions of the program. For a new version of the program, we compute the difference between the authorization constraints implied by that candidate placement in the new version and the authorization constraints for a previously validated hook placement. Programmers only have to review the differences in the authorization constraints between versions to validate the placement of the new version, as they have already accepted the authorization constraints that still apply. Refer to Appendix I for details about applying the approach to ten versions of the X Server program, which have had authorization hook placements for over ten years. We find that on average the programmers only need to examine 22 authorization constraints per version to resolve whether the authorization hook placement is correct, with a maximum of 42 constraints for version 1.16. Compared to identifying security-sensitive operations manually, this is a modest overhead in many cases.

REFERENCES

[1] J. Carter. Using GConf as an Example of How to Create an Userspace Object Manager. *2007 SELinux Symposium*, 2007.

[2] D.Walsh. Selinux/apache. http://fedoraproject.org/wiki/SELinux/apache.

[3] D. Muthukumaran, T. Jaeger, and V. Ganapathy. Leveraging "choice" to automate authorization hook placement. In *CCS'12: Proceedings of the 19th ACM Conference on Computer and Communications Security*, page TBD. ACM Press, October 2012.

[4] D. Muthukumaran, N. Talele, T. Jaeger, and G. Tan. Producing hook placements to enforce expected access control policies. In *Engineering Secure Software and Systems - 7th International Symposium, ESSoS 2015, Milan, Italy, March 4-6, 2015. Proceedings*, pages 178–195, 2015.

[5] G. C. Necula, S. McPeak, S. P. Rahul, and W. Weimer. Cil: Intermediate language and tools for analysis and transformation of c programs. In *Proceedings of the 11th International Conference on Compiler Construction*, CC '02, pages 213–228. Springer-Verlag, 2002.

[6] SE-PostgreSQL? http://archives.postgresql.org/message-id/20090718160600.GE5172@fetter.org, 2009.

[7] S. Son, K. S. McKinley, and V. Shmatikov. Fix Me Up: Repairing Access-Control Bugs in Web Applications. In *ISOC Network and Distributed System Security Symposium (NDSS)*, 2013.

[8] L. Tan, X. Zhang, X. Ma, W. Xiong, and Y. Zhou. AutoISES: Automatically inferring security specifications and detecting violations. In *Proceedings of the 17th conference on Security symposium*, pages 379–394. USENIX Association, 2008.

[9] Implement keyboard and event security in X using XACE. https://dev.laptop.org/ticket/260, 2006.

[10] X. Zhang, A. Edwards, and T. Jaeger. Using CQUAL for static analysis of authorization hook placement. In *Proceedings of the 11th USENIX Security Symposium*, pages 33–48, August 2002.

I. APPENDIX

HEIMDAL is written using the CIL framework [5] and all our code is written in OCaml. We present the results of applying HEIMDAL to X Server versions from 1.6 to 1.17. In Table I, we compare the number of automated hooks computed and manual hooks placed for corresponding X Server versions. In examining these results, we see that HEIMDAL recognizes the significant change between versions and 1.11 and 1.12, which resulted in the addition of 27 manual hooks, and identifies 28 new automated hooks accordingly. The automated method appears to be sensitive to other changes in the program, as the addition of new control statements or statements that obtain data using untrusted external inputs will create new security-sensitive operations with the potential of requiring new hooks.

Table II shows information about maintaining subsumption constraints across versions. The first column lists the total number of subsumption constraints in each version, and the two other columns list the number of new and modified subsumption constraints between versions. To compute these numbers, we removed the subsumption constraints that were unchanged between versions. An operation is defined by its object (variable), location (function), and structure member accesses dominated. As long as these are unchanged for two operations in the same subsumption constraint from the previous version, the constraint is unchanged.

We see that the most significant changes occur between versions 1.10 to 1.12 and versions 1.15 to 1.16. From 1.10 to 1.11 the programmers did not change the manual hooks, but modified the control flows under the hooks. Only three hooks saw changes in their structure member accesses, but the dominance relationships among automated hooks changed, resulting in new constraints (i.e., for new pairs of operations). Noting that the relationships have changed but they are still dominated by the same manual hooks may ease the programmers task in verification. For the transition from version 1.15 to 1.16, a significant number of variable names changed, which changes the operations and constraints. HEIMDAL displays type information with the constraints to enable programmers to see the cause of the change without having to scan unnecessary program information.

Similarly, Table III shows information about maintaining equivalence constraints across versions. The most significant number of new constraints occurs between versions 1.11 to 1.12, where much new code was introduced as described above. Very few equivalence constraints were modified across versions.

TABLE I
AUTHORIZATION HOOKS GENERATED FOR X SERVER VERSIONS

XServer versions	Automated Hooks	Manual Hooks
1.7	278	186
1.8	280	186
1.9	276	181
1.10	272	180
1.11	284	180
1.12	312	207
1.13	333	206
1.14	333	206
1.15	317	207
1.16	310	207
1.17	334	207

TABLE II
SUBSUMPTION CONSTRAINT RELATION ACROSS VERSIONS

XServer Versions Compared	Total Subsumption Constraints	New Constraints	Modified Constraints	% Effort
1.7 – 1.8	90	0	0	0
1.8 – 1.9	91	9	7	17.58
1.9 – 1.10	96	8	3	11.46
1.10 – 1.11	94	28	3	32.98
1.11 – 1.12	119	17	8	21.01
1.12 – 1.13	129	13	1	10.85
1.13 – 1.14	151	9	0	5.96
1.14 – 1.15	155	11	0	7.10
1.15 – 1.16	158	1	38	24.68
1.16 – 1.17	151	6	0	3.97

TABLE III
EQUIVALENCE CONSTRAINT RELATION ACROSS VERSIONS

XServer Versions Compared	Total Equivalence Constraints	New Constraints	Modified Constraints	% Effort
1.7 – 1.8	47	2	1	6.38
1.8 – 1.9	46	4	0	8.70
1.9 – 1.10	47	2	1	6.38
1.10 – 1.11	45	3	1	8.89
1.11 – 1.12	48	13	2	31.25
1.12 – 1.13	55	8	2	18.18
1.13 – 1.14	56	3	1	7.14
1.14 – 1.15	56	6	2	14.29
1.15 – 1.16	55	2	1	5.45
1.16 – 1.17	54	3	0	5.56