POSTER: On the Effect of Security Warnings on Cryptographic API Misuse

Peter Leo Gorski, Luigi Lo Iacono, Yasemin Acar*,
Sebastian Moeller†, Christian Stransky*, Sascha Fahl**
Cologne University of Applied Sciences, *Leibniz University Hannover,
†Technische Universitaet Berlin, **Ruhr-University Bochum

***Abstract*—Cryptographic API misuse is responsible for a large number of software vulnerabilities. In many cases developers are overburdened by the complex set of programming choices and their security implications. Past studies have identified significant challenges when using cryptographic APIs that lack a certain set of usability features (e. g. easy-to-use documentation or meaningful warning and error messages) leading to an especially high likelihood of writing functionally correct but insecure code.**

To support software developers in writing more secure code, this work investigates a novel approach aimed at these hard-to-use cryptographic APIs. In a controlled online experiment with 53 participants, we study the effectiveness of an API integrated security advice which informs about an API misuse and places secure programming hints as guidance close to the developer. This allows to address insecure cryptographic choices including encryption algorithms, key sizes, modes of operation and hashing algorithms with helpful documentation in the guise of warnings. Whenever possible, the security advice proposes code changes to fix the responsible security issues. We find that our approach significantly improves code security. 73% of the participants who received the security advice fixed their insecure code.

I. Introduction

A large number of software vulnerabilities is caused by developers who misuse security APIs [2], [5]. Previous work identified multiple trouble spots including secure network connections [3], the use of permissions in mobile apps [4] and the use of cryptographic APIs [1]. Some of the most serious data breaches in recent history were caused by not properly using TLS to secure data in transit or not securely storing data in rest. Such incidents affect millions or even billions of users worldwide and jeopardize their security and privacy.

While there is previous work that tries to improve code security by enhancing API simplicity or by providing IDE plugins [6], we propose a different and novel approach that allows providers of existing and future cryptographic APIs to improve code security. Therefore, they do not have to change their programming interfaces, rely on the development and integration of plugins for integrated development environments (IDEs) or hope that security of unsafe information sources such as Stack Overflow becomes better. Instead, we propose the integration of effective security advice directly into cryptographic APIs. We develop an API integrated security advice concept that provides context sensitive help and offers ready-to-use and secure code snippets to fix security issues. We implement our approach for Python and the PyCrypto cryptographic API and conduct a between-subjects online study with 53 experienced Python developers. In the course of this study we try to answer the following research question:
RQ: *Does the API integrated security advice have a significant effect on code security?* With this research question we try to assess the ability of our approach to improve code security. We analyze all changes made to the code after a security advice has been shown. We find that our approach had a significant positive impact on 73% of our participants who left their code insecure at the first place: They upgraded bad cryptographic choices to secure ones.

We find that similar to high quality developer documentation, good API design and helpful IDE plugins, our approach has a significantly positive effect on code security, but allows API providers to improve code security for existing cryptographic APIs in a bottom approach without having to change API design or relying on third party tools.

II. Security Advice

We implemented the security warning concept from above for a subset of the PyCrypto API for the Python programming language. Figure 1 shows a sample security warning for the insecure RC4 algorithm for symmetric encryption. To assess API call security, we followed the classification provided by Acar et al. [1].

The PyCrypto patch hooks specific API calls that create instances of weak cryptographic objects such as the call to `Crypto.Cipher.ARC2.new()` which creates a new cipher object that uses the insecure ARC2 algorithm. Whenever an insecure cryptographic object is created, our patch calls an advice method that uses contextual information to show a security warning.

III. Developer Study

We designed an online, between-subjects study to compare how effectively developers could write correct, secure code using either PyCrypto as a control, or our patched version of PyCrypto with the security intervention. We recruited 53 developers with demonstrated Python experience (on GitHub) for an online study; we also recruited via mailing lists and developer forums.

Participants were assigned to complete a short set of programming tasks; they were randomly assigned either the control condition that replicated the PyCrypto condition of the Acar et al. 2017 study on cryptographic Python APIs [1],

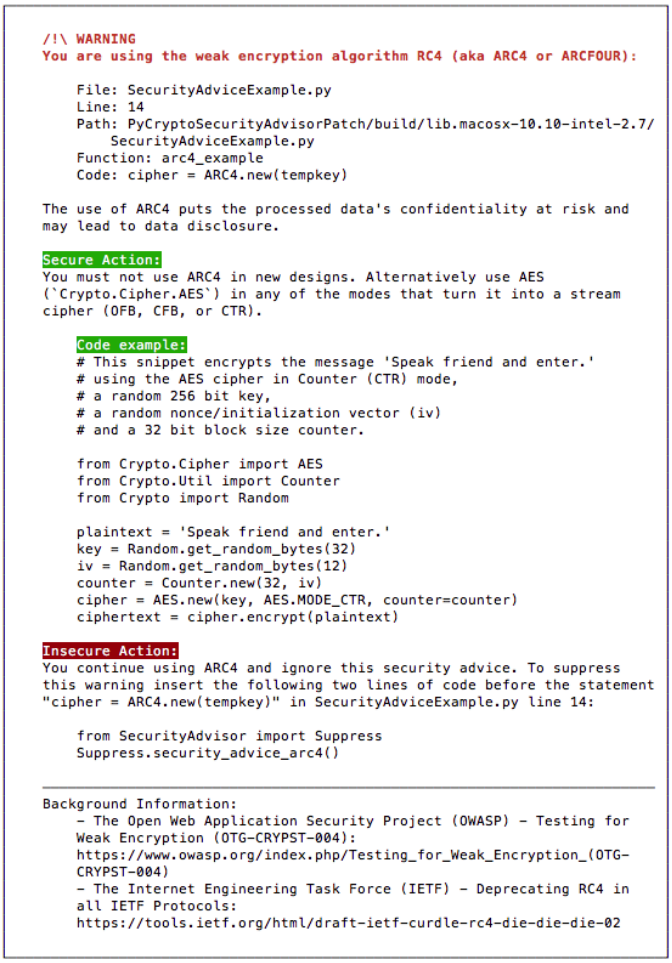

Fig. 1: Security advice for PyCrypto triggered by an RC4 usage and displayed in a terminal running python code.

or the PyCrypto patch condition, where we tested our security warning. All participants were given a symmetric encryption task and a symmetric key generation and storage task. We examined participants' submitted code for functional correctness and security.

Due to the location of our universities, there was no formal IRB process. We did, however, model our study material and procedures after an IRB-approved study and adhered to the strict German data and privacy protection laws.

IV. Results

Participants were generally successful in functionally solving the tasks, while security results varied across conditions, the patched condition being an improvement over PyCrypto where applicable. This improvement was pronounced: participants who wrote code that triggered a warning message were 15× as likely to convert it to a secure condition as opposed to participants who wrote similar insecure code in the PyCrypto condition.

We observed 26.9% secure solutions in the PyCrypto condition; compared with 50.7% in the PyCrypto patch condition. We were not able to obtain a meaningful regression

		Secure F	Secure T
Warning	F	21	1
	T	3	8

TABLE I: Contingency table for secure task solutions and triggered warnings used in our Fisher's exact test.

Factor	O.R.	C.I.	*p*-value
Warning displayed	4.70	[0.04, 492.14]	0.515
Storage Task	0.13	[0.01, 1.25]	0.078
Development experience	1.11	[0.88, 1.39]	0.378

TABLE II: Results of the final logistic regression model examining whether displayed warnings improve task security in cases where a warning would have been triggered. Odds ratio (O.R.) indicates relative likelihood of a task being secure.

model, caused by the small number of tasks that triggered and ended up with insecure code in the PyCrypto patch condition (11), as well as the small number of tasks that would have triggered a warning but were not modified to be secure in the PyCrypto condition (22). We followed this inconclusive model up with Fisher's exact test (cf. Table I, which was significant ($p<0.01$), with an odds ratio of 56. The warning messages were noticed by participants who saw them, which was clear both from self-reported memory of them as well as changes in their code: the warning message lead to a change from initial insecure code to a secure solution in most cases (8 out of 11). Generally, the applicability of the warning message was limited; it applied to 24 of 44 insecure solutions across conditions, and was shown in 11 of 22 insecure cases in the PyCrypto patch condition.

References

[1] Y. Acar, M. Backes, S. Fahl, S. Garfinkel, D. Kim, M. L. Mazurek, and C. Stransky. Comparing the usability of cryptographic APIs. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 154–171, 2017.

[2] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel. An empirical study of cryptographic misuse in android applications. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 73–84, New York, NY, USA, 2013. ACM.

[3] A. P. Felt, A. Ainslie, R. W. Reeder, S. Consolvo, S. Thyagaraja, A. Bettes, H. Harris, and J. Grimes. Improving ssl warnings: Comprehension and adherence. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 2893–2902, New York, NY, USA, 2015. ACM.

[4] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, pages 3:1–3:14. ACM, 2012.

[5] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov. The most dangerous code in the world: Validating ssl certificates in non-browser software. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 38–49, New York, NY, USA, 2012. ACM.

[6] D. C. Nguyen, D. Wermke, Y. Acar, M. Backes, C. Weir, and S. Fahl. A stitch in time: Supporting android developers in writingsecure code. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17. ACM, 2017.