Automatic Detection and Identification of Breaking Security-Configuration Rules

Master's Thesis

Supervisor: Prof. Dr. Alexander Pretschner
Advisor: Patrick Stöckle
Email: {alexander.pretschner, patrick.stoeckle}@tum.de
Phone: +49 (89) 289 - 17314
Starting date: immediately

Fakultät für Informatik
Lehrstuhl 4
Software & Systems Engineering
Prof. Dr. Alexander Pretschner

Boltzmannstraße 3
85748 Garching bei München

Tel: +49 (89) 289 - 17314
https://www4.in.tum.de

Context

Researchers and practitioners agree on the theoretical importance of security configuration: if we want a secure system, we have to configure it securely. Previous studies showed that *lack of knowledge and experience* are the main causes of security misconfiguration in the industry.[1] We can fight this problem of missing knowledge by using the guides published by the Center for Internet Security (CIS)[1] or the Defense Information Systems Agency (DISA)[2]. After solving the first problem, we have to deal with a second one which Dietrich et al. described as *Having other priorities*, *Time pressure*, and *Complexity of the System*: In the industry, we have complex systems using different programming languages and frameworks, running on different OSs, and communicating with various other components and systems. The companies rely on the systems running because they will lose money for every second the systems are not working. Even if the administrators knew how to securely configure the system under test, they would omit the hardening. To safely harden a running system, the administrator would have to know for every rule of the security-configuration guide that this rule is not interfering with the functionality of the system. We call such a rule interfering with the functionality of the system a **breaking rule**. It is not realistic that the administrators know for every security-configuration rule within a guide of hundreds of rules that this rule is not a breaking rule; if they would know the system by heart, they would not need the security-configuration guides of CIS and DISA and this would contradict the findings of Dietrich et al. Thus, there is a risk that there is at least one breaking rule within the guide. Therefore, the administrators will rather omit the hardening instead of risking to break the functionality.

Example An organization is using Windows 10 as OS for all employees. On the PCs, the employees use Microsoft Office to read and write emails and to create documents or tables. Thus, different workflows have been created in this organization that use the Excel macro feature, e.g., to calculate taxes based on the values in a given table. Nevertheless, the fact that macros are an essential part of the workflows in the organization is not known to all employees. The usage of macros should demonstrate in this case the problem of an implicit function of the system which is used in practice, but is not well documented.

Now, the system administrators of this organization were told by a colleague that the infrastructure would be more secure if it was hardened using a guide of the CIS. They apply all rules of the mentioned guide to a test machine with Windows 10, Office, and other tools used in the organization. Afterward, they test all the functions the system should fulfill but – as they are not aware of it – they do not test for the macros to work. After the – from their perspective – successful test, they apply the guide to all PCs in the organization.

Shortly after the rollout, the employees using the macros complain that their workflows are not working anymore; in other words, the employees have *detected* that there is a breaking rule. The system administrators now have two options:

- They sit down with the employees affected by the breaking rule. They try to understand what the employees want to do and why it is not working anymore. Then, they are going through the list of rules and search for rules which might break this functionality, e.g., in our case a rule stating that macros should be disabled. They disable the rule and test if the functionality is working again. If so, they have found the breaking rule for this functionality. If not, either the rule was affecting the functionality not at all or there are also other breaking rules. In this case, the administrators have to search for other rules. This whole process of *identifying* the breaking rules in the set of rules will take a lot of time and resources. As long as the rules are still in place, the employees cannot work or, at least, they cannot do the work related to those workflows.

[1] https://www.cisecurity.org/
[2] https://public.cyber.mil/stigs/scap/

- The administrators rollback all rules and reset the PCs to the state before the hardening. In this case, the functionality should directly work again, but we lose all the *security* gained through the hardening.

The *gains in security* are usually not as visible as the actual losses through the decreased productivity resulting from the broken workflows. Thus, most system administrators would opt for the second option which results also for them in way less work.

Even administrators who are very passionate about security will not this very often; at some point, they will leave the system untouched by any hardening measures and follow the proverb: *never change a running system*

This is the status quo of the hardening in the industry. In this thesis, we want to improve this situation by developing techniques to detect the breaking rules and identify them among the set of rules defined by a security-configuration guide.

Formalization In a more formal way, a functionality f can be expressed as a predicate of a system $s \in \mathcal{S}$ at a point in time t

$$f : \mathcal{S} \times T \rightarrow \{True, False\}$$

with the semantic that $f(s,t) = True$ means that the system s is providing the functionality f at t. A security-configuration rule r can also be expressed as such a predicate; $r(s,t) = True$ means in this context that the system s is compliant to the security rule r at t. Breaking b can then be expressed as a predicate of a security-configuration rule r with respect to system s, a point in time t, and a given set of functionalities $\mathcal{F}$.

$$b_{s,t,F}(r) = True \Leftrightarrow ((\forall f \in \mathcal{F} : f(s,t)) \wedge \neg r(s,t) \wedge r(s,t+1)) \Rightarrow \exists f \in \mathcal{F} : \neg f(s,t+1)$$

Non-interfering n can be defined for a set of rules $\mathcal{R}$ via

$$n_{s,t,F}(\mathcal{R}) \Leftrightarrow (\forall f \in \mathcal{F} : f(s,t)) \wedge (\forall r \in \mathcal{R} : r(s,t+1)) \Rightarrow (\forall f \in \mathcal{F} : f(s,t+1))$$

For a given security-configuration guide $\mathcal{G} = \{r_0, \ldots, r_n\}$, we want to calculate a subset $\mathcal{G}^*$ with the following properties

- $\mathcal{G}^* \subset \mathcal{G}$
- $n_{s,t,F}(\mathcal{G}^*) = True$, i.e. $\mathcal{G}^*$ is non-interfering.
- $\mathcal{G}^*$ is maximal, i.e. $\forall r \in \mathcal{G} \setminus \mathcal{G}^* : \neg n_{s,t,F}(\mathcal{G}^* \cup \{r\})$. Any additional rule added to $\mathcal{G}^*$ would break at least one functionality. With this criterion, we want to exclude algorithms which return always the empty set $\emptyset$.

Goal

This thesis is designed to tackle the following problems

- The administrators responsible for the hardening of the system are usually not the developers of the system. They do not know every functionality the system should provide and, therefore, it is difficult for them to assure that everything is still working after the hardening.
- If the administrators detect that the application of a security-configuration guide breaks certain functionality, they do not know which rule was the breaking rule.
- If the administrators identified one rule of the guide as a breaking rule, they have to check the functionality of the system again after all rules except for the breaking rules have been applied; maybe the guide includes more breaking rules. In the end, the

To overcome these problems is the goal of this thesis. As Windows 10 is still the OS used the most for PCs in companies and organizations, this work will focus on Windows 10 and Microsoft Office security-configuration guides.

Fakultät für Informatik
Lehrstuhl 4
Software & Systems Engineering
Prof. Dr. Alexander Pretschner

Boltzmannstraße 3
85748 Garching bei München

Tel: +49 (89) 289 - 17314
https://www4.in.tum.de

Working Plan

1. Literature: Can we use existing approaches from the regression testing, test automation, and security configuration domain to solve our problem? Especially literature regarding smoke tests might be helpful.
2. Design: How can users, e.g., the employees from above, define tests for the functionality so that we can execute them automatically? How can we create an environment so that we can apply the rules and the tests? After detecting the presence of breaking rules, how can we find them efficiently? How do we deal with the fact that sometimes only the combination of two or more rules breaks a functionality?
3. Implementation: Within this thesis, we want to implement a proof-of-concept implementation of the developed approach. The main focus lies on the application of Windows-related guides. Thus, our PoC should focus on Windows 10. The use of virtualization technologies like VirtualBox[3] or Vagrant[4] and provisioning technologies like Ansible[5] might be helpful.
4. Evaluation: The evaluation could be the following. We take a guide, e.g., the CIS Windows 10 guide. We define a set of test ensuring that our functionality is preserved. Most of them are not affected by the guides, but some tests should be broken by one or more rules. Maybe, we could also introduce some tests which cannot work even if the rules are not applied. The software should then
   - detect that there are breaking rules in the security-configuration guide regarding the given tests
   - identify the rules that are breaking rules
   - for each failing test identify which rules were causing this test to fail
   - determine a maximal, non-interfering subset of rules.

References

[1] Constanze Dietrich, Katharina Krombholz, Kevin Borgolte, and Tobias Fiebig. Investigating system operators' perspective on security misconfigurations. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1272–1289, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5693-0. doi: 10.1145/3243734.3243794. URL http://doi.acm.org/10.1145/3243734.3243794.

Fakultät für Informatik
Lehrstuhl 4
Software & Systems Engineering
Prof. Dr. Alexander Pretschner

Boltzmannstraße 3
85748 Garching bei München

Tel: +49 (89) 289 - 17314
https://www4.in.tum.de

[3] https://www.virtualbox.org/
[4] https://www.vagrantup.com/
[5] https://www.ansible.com/