Securing Secrets in Git: Analyzing Ansible Vault and Git-based Vault File Management

Ingmar Fjolla
ifjolla@ncsu.edu

Abstract

In an executive order released by president Biden, the critical role played by enterprises in fortifying the nation's cybersecurity was emphasized. The 2022 state of enterprise open source report released by Red Hat states that 80 percent of surveyed IT leaders expect to increase their use of enterprise open source software for emerging technologies. Ansible is an open source Infrastructure as Code tool that is used, among other things, for security automation. Ansible includes a tool, Ansible Vault, to safely store encrypted secrets in Git along with the source code. Because Ansible is used in automation tasks that are considered mission critical, obtaining passwords to these resources and compromising them can be devastating for an organization. This paper presents an in depth review of the state of security and usability in the Ansible ecosystem. We analyze various online platforms to find where practitioners struggle with Ansible Vault in terms of usability and categorize the top 4 issues. The paper analyzes the security of the implementation and identifies potential side channel attacks in the open source supply chain that can be generalized for other open source projects. The motivation for this paper is not just to showcase potential attacks, but to contextualize why attackers might go through the effort to compromise a team's Ansible Vault.

1 Introduction

Ansible is an extremely popular and powerful open source tool used for automation. Some examples of the way Ansible is used include configuration management, cloud provisioning, network automation, and security-related tasks[1]. In some of the public case studies that Ansible publishes[2], we see companies like NASA, US Government contractors, British Army, and some financial institutions using Ansible. Since these are just public customer references, it is safe to assume many more use it with over a quarter million[3] downloads a month. With that in mind, it's more important than ever to look into what attacks can happen and how to keep yourself secure when using Ansible.

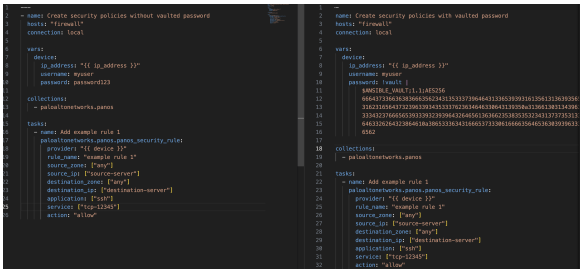

Figure 1: Vaulted password example

This paper aims to answer the following research questions:

RQ1: *How do the current attacks against Ansible Vault work to retrieve the key? (and can we contribute a fix against this)*

RQ2: *Are there other ways to obtain the key or plaintext other than the attacks that currently exist?*

RQ3: *Are there mechanisms in place that Ansible Vault has to make it more secure and are developers taking advantage of these added bits?*

RQ4: *Where do developers struggle with Ansible Vault?*

At a high level, Ansible works through the concept of control nodes and managed nodes. A control node is defined as the machine Ansible is installed on, and managed nodes are devices that is going to be managed by that control node. A developer/practitioner will then create a playbook[7] specifying what they want the desired state to be and run that playbook to perform the actions specified in the playbook. Because some of these playbooks contain sensitive data (i.e passwords), Ansible provides a tool, ansible-vault[6] (also referred to in this paper as Ansible Vault), to encrypt the sensitive data at rest. This allows developers to safely store their source code (playbooks,inventory files) in version control. An example is seen in Figure 1 of a playbook that automates security pol-

[1] https://www.redhat.com/en/technologies/management/ansible/use-cases

[2] https://www.ansible.com/resources/case-studies

[3] https://www.ansible.com/overview/it-automation

icy creations for PAN-OS[4] devices before and after vaulting the password. This can be done at the variable level as seen in the previous figure, or at the file level to encrypt an entire file that contains sensitive data. Because Ansible is used in security and network automation tasks that are considered mission critical, this the vaulted values high value targets for adversaries attempting to obtain passwords to these resources.

Previous research on the security of Ansible has primarily focused on user-related code smells [40] [45] [46] or the security use cases of Ansible. Our research shifts the focus towards a comprehensive analysis of Ansible Vault's usability and examination of attack vectors. We showcase a potential stealthy attack that falls under the category of **Subvert Legitimate Package** in Ladisa et al's.[33] taxonomy of attacks on the open source software supply chains. Our analysis also covers the main gripes users have with the tool, and what are the most common struggles. We aim to contribute a different perspective to the field, enriching the dialogue around open-source tools beyond the emphasis on user errors. In doing so, we highlight the need for more research on the topic of authenticating open-source artifacts.

2 Overview

This section details an overview of our approach and methodology in answering the research questions, along with the relevant threat models necessary to answer **RQ1** and **RQ2**.

In the context of enterprise, the trusted computing base can be quite large depending on what an organization's setup is. For this paper, the trusted computing base includes the version control repository, the Ansible package, the dependencies of the Ansible package, containers, Pip/Python, and any internal repositories for packages that an organization might make use of to mirror external packages (internal image registry or python package registry for example). If an organization is using Ansible Automation Platform, that adds another potential attack vector as well if permissions are not configured correctly. However, we will not consider attacks on the control node.

The goal of these attacks is to get the data that is encrypted in the vault. This is done through offline cracking attacks, compromised supply chain attacks (which includes both the chosen nonce attack and malicious code injection), and leveraging unused authorization of credentials on Ansible Automation Platform(AAP).

For all attacks in this paper, we are assuming there is at least one compromised individual with at minimum access to a private git repository (where Ansible playbooks and resources are stored) inside a company/agency. The attacks also vary in level of involvement of the attacker(s) and we will start with the most accessible attack and then move up to the attacks that rely on compromising the open source supply chain.

2.1 Research Question 1

2.1.1 Methodology

RQ1 aims to understand the way current attacks against Ansible Vault work, and how we can contribute upstream to Ansible to either protect against these attacks, or make their execution more expensive. Since the current attacks use open source tools like Hashcat[5], we perform code reviews on the cracking attack, and Ansible Vaults implementation to better understand the mechanics of the attack. We look at the history of the tools to see what changes may have been made in the last couple of years. Our pull request will aim to be usable, and minimally invasive in the context of Ansible's code base by using the same cryptography library that Ansible uses and following the contributor guidelines. Finally, we collect sources/walkthrough's of the attacks along with characterizing the accessibility of these tools in the era of GPU speed we are at today.

2.1.2 Threat: Offline Cracking Attacks

This attack involves using popular password cracking tools like Hashcat or John The Ripper[6] (though this paper primarily focuses on Hashcat). Our assumptions are the attacker has read access to a repository for Ansible playbooks and the vault file(s). Although the organizations version control repository might be private, if they haven't locked down the repository to just their team anyone in their organization might still have access to this. The adversary could be working alone, or as part of a group of cyber criminals/nation state. The adversaries will be well funded, however even an adversary who is just experimenting on a weaker machine would be able to attempt this attack (without much success, as we showcase later).

[4] https://pan.dev/ansible/docs/panos/tutorials/config-secpol-and-objs/

[5] https://github.com/hashcat/hashcat

[6] https://github.com/openwall/john

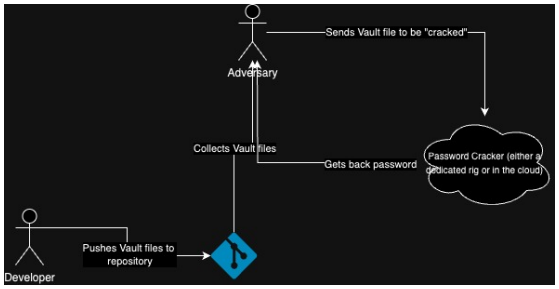

Figure 2: Flow for Offline Cracking Attacks

2.2 Research Question 2

2.2.1 Methodology

RQ2 aims to find other potential attacks that may exist to extract the data in the vault. We do this through code analysis again, and by finding side channels. In the context of this paper, our side channel will be attacking the open source supply chain so we will look at ways to compromise the trusted computing base which is described in the evaluation section to see what components can be compromised. While these attacks are theoretical (as in no documented cases of this attack occurring), we will use previous examples of attacks in the software supply chain[33] along with showcasing our experiments and documenting what issues we find along the way. Finally, we will identify gaps we've found and lay the ground for future work that we want to pursue in open source supply chain security.

2.2.2 Threat: Compromised Supply Chain Attacks

This attack is makes more assumptions about an attackers privileges rather than their hardware or monetary capabilities. We assume that the compromised individual within an organization may be apart of a platform team[7] or something similar allowing them to compromise the supply chain for developers. However, we will also make the case that this could happen external to an organization in the evaluation section. In this example, our assumptions are that a team using Ansible (let's call them the automation team) might reach out to the platform team for resources like containers/virtual machines or to mirror external packages in an internal repository for compliance. The adversary can "bake" environment variables (or files) into these containers and if teams don't perform their own compliance checking after the fact, it can leave them susceptible to attacks. Two examples that we show in our experiments later are the **ANSIBLE_VAULT_ENCRYPT_SALT** variable which forces Ansible Vault to reuse the salt each time leaving them susceptible to chosen nonce attacks. And the **PIP_INDEX_URL** variable which will point to another index for python packages. For the pip environment variable, we demonstrated how we didn't need to poison Ansible or all of it's dependencies for this attack to work. Rather, we only had to poison the cryptography library.

[7] https://martinfowler.com/articles/platform-teams-stuff-done.html

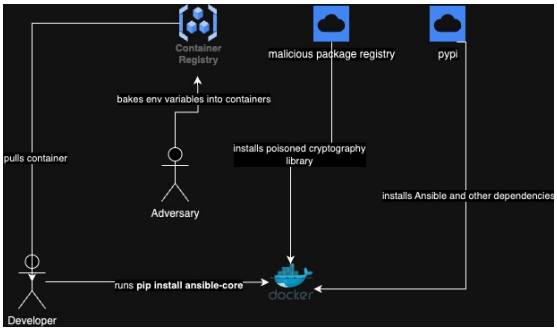

Figure 3: Example flow for receiving poisoned dependencies

2.3 Research Question 3 and 4

RQ3 and **RQ4** will follow similar methods, so we will group them together. To answer the question of whether Ansible Vault has other mechanisms to make it more secure and the overall usability, we perform extensive documentation review to see what we can find on these topics. Our goal is to see what advice is out there in the form of gray literature, and see what added layer of security the solutions may offer. The Ansible documentation makes mention of some different avenues for asking questions aside from Stack Overflow, so we will also attempt to group some of the more commonly asked questions on those channels (mostly Google Groups) along with general Github Issues. We collect internet artifacts using different search strings which we discuss later in the paper. Characterizing trust is difficult, so we hope to find different examples of both criticism of Ansible Vault as well as examples of support with best practices. Finally, we will also try to quantify the amount of amount of public vault's that might make use of the added security mechanisms we may find.

3 Current Attacks on Ansible Vault

This section details answers to **RQ1**, along with potential mitigations and experiments relating to the attack.

3.1 Background

Firstly, we had to identify whether or not there were any ways to "crack" the vault in its current state. We

found several[25][50] articles online detailing specialized modes in both Hashcat and John the Ripper that retrieve the key that was used to encrypt the vault. Interestingly, we have not seen this attack mode mentioned in the Ansible specific forums. When we were performing our literature review to answer **RQ3** and **RQ4**, when the topic of overall security of ansible-vault comes up[12][51] users generally refer to the strength of AES, and don't mention the password based key derivation function (using SHA256) and how specialized hardware can attack that.

3.2 Approach

To answer **RQ1**, we start with the implementations of both attacks[27][30] from Hashcat and John the Ripper along with the actual implementation[8] of Ansible Vault. When ansible-vault is invoked, it first generates a random salt (if one is not given). Then, it uses a password based key derivation function (PBKDF) with SHA256 as its hash algorithm to derive a password for the encryption, authentication, and an initialization vector(IV). The first derived password and the IV are used for the AES CTR encryption to generate the ciphertext, and then the authentication password to compute an HMAC over the ciphertext (encrypt then mac). Then the salt, HMAC, and ciphertext are concatenated and hexlifyed to create the final vault[6].

The attacks focus on the key derivation function by computing hashes and then comparing a portion of the output digest to the one that the vault used for its authentication. Users first use a John the Ripper script(ansible2john)[31] to convert the vault to a format ready for either John the Ripper or Hashcat to use. Our focus was on Hashcat, which has a dictionary attack. This allows users to specify a wordlist of passwords that it can loop through and calculate. There are other attack modes that we mention later, but our experiments just focused on that.

3.3 Evaluation

What makes this attack dangerous is the speed at which these tools can calculate hashes, along with accessibility of these tools to potential adversaries. In an example benchmark[16] posted by a maintainer of Hashcat, the RTX 4090[8] GPU was able to compute around 912,000 ansible-vault hashes per second.

In our experiments, we ran Hashcat in a constrained Kali Linux[9] Docker container with 4gb of ram allocated to it, and found that in a traditional dictionary attack it took about an hour to go through the entire rockyou wordlist. However, we randomly chose 10 public vaults and doing this attack was unsuccessful because the passwords the users had for their vault are not in the wordlist. We did not attempt a brute force or masking attack, however we found a model online [22] called money to crack which quantifies the cost of an attack which we used to evaluate the cost of brute forcing an ansible-vault hash.

On average, the RTX 4090 retails for $2,000 and over the course of its lifetime(7 years in this scenario) will incur an energy cost of $2,761.29 if we assume 0.45 kWh of energy consumed an hour. We can then estimate that the GPU can compute around 46.19 billion ansible-vault hashes per dollar over its lifetime. For a 10 character password with only uppercase and lowercase characters, that brings us to an average cost of $1,708,171 to crack. This is theoretical and does not guarantee that an adversary who invests that much is guaranteed to crack the password, rather it provides a way to quanitfy the cost of cracking as an alternative to time to crack.

One answer to this problem is to follow general advice on creating longer passwords, not reusing passwords across sites, and other recommendations from researchers about password strength. However, in an attempt to make it harder for a potential adversary to crack an ansible-vault, we opened a pull request[28] that would have enabled support for Scrypt[41] as a key derivation function. In the same benchmark used above on the 4090 GPU, Hashcat is only able to calculate around 7156 hashes per second which is a considerable slowdown.

Users would have been able to choose Scrypt through means of an environment variable, however the pull request was closed because there was not much community interest at the time of writing, and more modifications would have been needed that the maintainers did not think was worth the effort. We asked in the Ansible matrix[10] chat what we could do to get it merged in, but our comments did not recieve much interest.

4 Potential Attacks on Ansible Vault

This section details answers to **RQ2**, along with potential mitigation's and experiments relating to the attacks.

4.1 Background

Because we were not able to attack the implementation of Ansible Vault itself aside from the attack shown in the previous section, we had to look for ways to extract the key or plaintext through other side channels as discussed in section 2.2. Ladisa et al.[33] created a comprehensive list of attack vectors on the open source supply chain,

[8] https://www.nvidia.com/en-us/geforce/graphics-cards/40-series/rtx-4090/

[9] https://hub.docker.com/r/kalilinux/kali-rolling

[10] https://app.element.io/#/room/#devel:ansible.com

where our main focus in this paper falls under the category of **subvert legitimate package**. The assumption is that a developer may be using a container from a public registry that has environment variables baked into it that aim to compromise the security of the Ansible package, though this attack can also work on other Python packages as well. Recent research[36] has shown successful examples of attacks with containers with almost 17,000 downloads where malicious images hosted on Docker used typosquatting to impersonate trusted images, only to infect users with crypto-miners.

Traditionally developers using Ansible may not have been performing their development using containers. However, virtual environments have been replaced with execution environments[21] with the addition of projects like ansible-navigator[11] and Ansible Automation Platform[12].

4.2 Approach

The first environment variable we exploited was the **PIP_INDEX_URL** variable, which is a feature of pip allowing users to specify an alternate registry. We built a Python container using the official Python Docker image[19] and included a line in our Dockerfile exporting our malicious package repository as the **PIP_INDEX_URL** variable. In our scenario, when a user runs 'pip install ansible-core', the official Ansible package is installed from pypi.org along with every dependency except for the cryptography[17] package, which gets downloaded from our malicious package repository. We hosted the malicious repository locally using Docker as well, but an attacker can easily host their own malicious repository available on the internet.

Our tampered package[23] sends a post request to another malicious server we were hosting every time the PBKDF function in the cryptography library is called. So when a user encrypts a variable with ansible-vault, our server receives the password the user used for that operation. Traditional image scanners would not pick this up as a potential vulnerability because it's only an environment variable, so if users don't inspect the container manually they would not know that this variable was set.

Another variable that one can exploit is the **ANSIBLE_VAULT_ENCRYPT_SALT**, which allows users to set their own salt for ansible-vault to encrypt their variables with. Although this might be purposefully set by users, an attacker can bake this into a Dockerfile as well. The attack is only successful if a user uses the same key to encrypt two different vaults, leaving them susceptible to chosen plaintext attacks because the initialization vector is now the same.

[11] https://github.com/ansible/ansible-navigator

[12] https://www.redhat.com/en/technologies/management/ansible

Since the result of performing an xor operation on two ciphertexts (vaults in this case) reveals information about the plaintext, an attacker can employ different techniques to attain the information in the vault. They would not find out the password, just the information in the vault. In the original Github issue[5] requesting the feature, we found that users requested a way to set this because every time a file or variable is decrypted then re-ecnrypted Git thinks that the file changed since the ansible-vault implementation generates a new salt each time. So in order to work around this, users can supply their own salt through the environment variable to make the encryption more deterministic.

4.3 Evaluation

To evaluate how developers can protect themselves, we used pip secure installs[20] as a potential technique to ensure developers receive authentic packages. Developers would need to make modifications to the way they run the pip install command. For example, instead of `pip install ansible-core`, a developer would need to create a requirements file like this:

```
ansible-core==2.16.0 --hash=sha256:...
MarkupSafe==2.0.0 --hash=sha256:...
jinja2==3.1.2 --hash=sha256:...
PyYAML==6.0.1 --hash=sha256:...
resolvelib==1.0.0 --hash=sha256:...
cryptography==41.0.4 --hash=sha256:...
packaging==23.2 --hash=sha256:...
```

where the developer has to specify a specific version and hash that they can find on pypi for the specific package. This technique works because the tampered cryptography package hash on our malicious repository would not match with the authentic one, so we attempted to circumvent this. We modified the backend code[18] of our package repository to show the authentic hash throughout. However, when the package gets placed on a developers machine pip would error out saying that the expected hash does not match the actual hash. We did note that if a poisoned dependency was already installed on a machine, that the secure hashing does not work because it sees that a package with the same name and version is already installed. So developers should take note about what packages are installed already, which can be tricky if they work with many dependencies and packages.

5 Ansible Vault Usability

This section details our answers to both **RQ3** and **RQ4**.

5.1 Background

A fundamental goal of tools that involve security is to not only ensure they are robust against attacks as we evaluated in previous sections, but to ensure they are accessible and usable by developers. An important part of evaluating this usability is looking to online sources and understanding where developers struggle, and what advice is usually given. Previous research has used mixed method studies involving developer interviews and or extensive reviews of online resources such as forums, articles or GitHub issues[3][38][11][2]. Our research focuses on the latter.

5.2 Approach

To answer **RQ3**, we did a deep dive into the official documentation and performed search queries looking for developer tutorials on using ansible-vault. We specifically used search strings including: ansible-vault tutorials, ansible-vault hardening, ansible vault guides. We collected various guides and excluded material from our list that were similar to other guides.

For **RQ4**, depending on the platform our methodologies were different. On the official Ansible GitHub repository we collected all issues that contained "vault" in the title or body. We did not include any exclusion criteria because we wanted to include a wide range of user interactions and experiences even if they were not technically usability issues. We then categorized the top issues we saw. However on other mediums like the Ansible google group, Stack exchange, or Reddit we manually filtered through user's posts on ansible-vault and collected the three most common groupings of questions that we observed.

5.3 Evaluation

There aren't any ways to make ansible-vault more cryptographically secure in the traditional sense as far as we can tell at the time of writing. For example users can't update the iteration count for the PBKDF or use different algorithms (unless they manually change the Ansible code). However, developers can make use of 2 features of ansible-vault:

1. Vault id's
2. Vault password client scripts

which we categorize as security features. A vault id is an identifier for a vault secret, allowing developers to distinguish between vault passwords when they are using multiple. Using multiple passwords for different vaults is desirable, so that if one vault password is leaked the other vaults are still secure. To quantify how many developers might be making use of this feature, we collected 1406 public vaults from GitHub and found that 1378 were using vault id's. Vault password client scripts allow users to dynamically retrieve their vault password, however quantifying this proved difficult since there wasn't a unique way for us to determine whether a file was a client script.

To begin our evaluation of where developers struggle with Ansible Vault, we started with categorizing the top 10 topics of discussion in the Ansible Github Issues using Latent Dirichlet Allocation(LDA)[13] for the topic modeling. We found duplicate topics of disucssion, which might be because we did not do any fine tuning and are taking the models at face value, so we grouped the topics into 4 broader categories:

1. SSH and security
2. Configuration and environment setup
3. File handling and Python package issues
4. Vault and encryption-related issues

Most of the GitHub issues were bug reports that might include ansible-vault as a side effect, and we found were not necessarily related to usability of ansible-vault as seen in the categories we showed.

For our forum review we followed a similar approach to[10] called open coding, to perform a qualitative review on different posts. We found the 4 most common topics that developers would inquire about were:

1. Multiple password support
2. Programatically getting the vault password
3. Password distribution
4. Best practices

6 Related Work

To better structure the presentation of related research, given the broad range of topics covered in this study, we have divided the Related Works section into different subsections to better maintain content cohesion. We also illustrate how our work differs and what context the related research may provide.

6.1 Infrastructure As Code

Previous security research on Ansible as an Infrastructure as Code (IAC) tool has primarily focused on its use cases for security [4][9], or on the security of the playbooks themselves. In one of their initial studies, Rahman et al investigated gaps in research related to IAC, and categorized the available research into four topics: framework/tools for infrastructure as code, adoption of infrastructure as

code, empirical studies, and testing[43]. They found that at the time of writing, no security research on the IAC frameworks or scripts themselves was being done. In subsequent research by Rahman et al. six security smells were defined specifically for Ansible scripts (but others were also defined for tools like Chef and Puppet) [44][45]. These smells included empty password, hard-coded secret, no integrity check, suspicious comment, unrestricted IP address, and use of HTTP Without SSL/TLS [45]. A static analysis tool specifically for Ansible, SLAC, was then created to lint Ansible scripts for those smells finding at least twenty five percent of analyzed scripts to contain at least one smell. Saavedra et al. created GLITCH [46] which aimed to be a language agnostic IAC static code analyzer to detect security smells. GLITCH was shown to outperform SLAC while using the same security smells. Opdebeeck et al. optimized security smell detection in Ansible scripts by utilizing Program Dependence Graphs (PDG's) that were context aware [39][40]. Their research showed that previous tools lacked awareness of Ansible specifc syntax and did not take into account control-flow or data-flow. According to their results, utilizing PDG's their linting tool, GASEL, outperformed the previous tools. None of the projects mentioned Ansible Lint[13], despite its popularity among practitioners with over 10,000 downloads a week[14], and an analysis of whether any security smell additions could have been added to the linter offered by the Ansible project. We will follow an "Always Contribute Back" approach and look for ways we can give back to the community if we identify improvements that could be made. Although our research will not be focusing on these linting tools or creating another one, our methodologies for collecting Ansible repositories will be similar. Our research will also primarily focus on Ansible Vault and consider what security smells exist in the context of the vault. Our research will also include a conversation on Ansible Navigator and Ansible Automation Platform (Ansible Tower) because as far as we know no previous research has been done in how these new tools affect the current threat model and whether any extra considerations should be made when considering an Ansible repository.

6.2 Secrets Management

Secrets management is a difficult topic that developers struggle with. One of the most thoroughly documented concerns is secret leakage, covering its extent, underlying causes, preventative measures, and remediation [1][32][35][49]. Basak et al. identified some of the challenges developers face with "Checked-in Secrets", and the answers they would receive on online forums when asking for help [10]. Followup research by Basak et al. and Rahman et al. details some best practices for secrets management [42][48]. Only two best practices specifically relating to Ansible Vault were found in the latter's research which were: the separation of the vault files from version control (VCS) and the use of vault id's. This shows a gap in research relating specifically to the overall usage of Ansible Vault and what developers may be struggling with. The research cited makes use of Stack Overflow[15] [10] and gray literature and we intend to expand our search queries to also include Github Issues, Red Hat documentation, and the Ansible Google groups[16] to evaluate Ansible Vault.

6.3 Open Source Software Security

In a qualitative study conducted by Wermke et al. where they interviewed individuals from various industry projects/roles, they found that all participants used Open Source Components (OSCs) in their projects and had a positive sentiment about their experiences with the Open Source Ecosystem [53]. Open Source Software Practices and Security are well researched topics [37][47] and a literature review by Shao-Fang Wen finds that 'Verification' is the most cited security category in Open Source research [52]. Interestingly, we could not find any academic papers related to verifying the security of the implementation of Ansible Vault (or any of the Ansible framework itself) [8]. As we noted before, most research concerning Ansible focuses on security smells in scripts or using Ansible for security purposes. The Ansible documentation explains their algorithms used [6] and in an article published by Gabriel Birke, [12] they note:

> As long as the vault password is a sufficiently long random password, it is very secure.

Our paper will be the first to verify the cryptography implementation of Ansible Vault and make use of previously published work on Python crypto misuses to aid in our findings [54].

6.4 Password Cracking

Prior research has shown the power of password recovery tools like John the Ripper(JTR) and Hashcat, along with more recent advancements in enhancing them [26][29][34]. Brogada et al. found that the Head and Tail technique [15] for hashing passwords could not be cracked by JTR or Hashcat [14]. These two tools were chosen because they have native support for cracking Ansible Vault files [27][30], and they are open source making them available for use by adversaries with low and high

[13]https://github.com/ansible/ansible-lint
[14]https://snyk.io/advisor/python/ansible-lint
[15]https://stackoverflow.com/
[16]https://groups.google.com/g/ansible-project

computing power. Although no academic research has been published on the specific mode for Ansible Vault, there have been multiple articles published that details its usage [25][50]. Our research will leverage this literature and look for public vault files and assess whether or not they are using secure vault passwords. We will also not be focusing on optimized techniques for the tools, rather we will use them as benchmarks to establish what might be feasible for different adversaries in the concept of Ansible Vault.

7 Discussion

In this section we talk about some of the implications and limitations of our work.

Implications: Through our experiments, we showed that there are ways adversaries can attain either the key, or data within the vault. However, even the least invasive attack requires high computing power and a high budget. Organizations should prioritize security training and awareness and consider things like the Supply-chain Levels for Software Artifacts[17] for securing their supply chain.

We also found some common questions and scenarios online that could be good candidates for enhancements in the Ansible documentation itself.

Limitations: Because of the nature of the qualitative analysis on the internet artifacts we studied, our findings might change on external validation and are subject to other interpretations. Our reproducer material is available here[24] for review.

8 Conclusion

Our research provided insight into the state of security and usability in the Ansible ecosystem with a focus on the ansible-vault tool. We provided a comprehensive analysis of the current vulnerabilities that affect ansible-vault and some potential ways adversaries can hijack the software supply chain to poison the dependencies of an authentic package while maintaining the authenticity of the original package a developer wanted to install.

While we investigated pip secure installs as a potential solution, future research can use our example attack to test the effectiveness of other software supply chain integrity solutions that are not language/ecosystem dependent. Our findings can also be generalized and other secret management tools can be investigated in the same way, by evaluating the security and usability, to assess the state of the secret management ecosystem. Future research can also make use of user experience surveys and interviews with developers to understand how not just ansible-vault is used in their ecosystem, but perhaps compare it to other secret management tools they use for infrastructure as code as well.

While our potential pull request upstream to Ansible may have been closed due to lack of interest/ not enough developer resources, future research should continue to interface with upstream developers to get feedback and understand ways we can always contribute back.

[17] https://slsa.dev/

References

[1] No need to hack when it's leaking. PDF document.

[2] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky. You get where you're looking for: The impact of information sources on code security. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 289–305, 2016.

[3] Y. Acar, C. Stransky, D. Wermke, M. L. Mazurek, and S. Fahl. Security developer studies with github users: Exploring a convenience sample. In *Thirteenth Symposium on Usable Privacy and Security, SOUPS 2017, Santa Clara, CA, USA, July 12-14, 2017*, pages 81–95. USENIX Association, 2017.

[4] R. Acheampong, T. C. Bălan, D.-M. Popovici, and A. Rekeraho. Security scenarios automation and deployment in virtual environment using ansible. In *2022 14th International Conference on Communications (COMM)*, pages 1–7, 2022.

[5] agowa. Issue 35480 on ansible/ansible. `https://github.com/ansible/ansible/issues/35480`, 2023. GitHub Issue.

[6] Ansible. Protecting sensitive data with ansible vault. `https://docs.ansible.com/ansible/latest/vault_guide/index.html`, 2022. Website.

[7] Ansible. Using ansible playbooks. `https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html`, 2022. Website.

[8] Ansible GitHub Repository. ansible/ansible - lib/ansible/parsing/vault/__init__.py. `https://github.com/ansible/ansible/blob/devel/lib/ansible/parsing/vault/__init__.py`. GitHub Code File.

[9] F. Araujo and T. Taylor. Improving cybersecurity hygiene through jit patching. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, page 1421–1432, New York, NY, USA, 2020. Association for Computing Machinery.

[10] S. K. Basak, Lorenzo Neil, Bradley Reaves, and L. Williams. What Challenges Do Developers Face About Checked-in Secrets in Software Artifacts? In *Proceedings of the IEEE/ACM International Conference on Software Engineering*.

[11] S. K. Basak, L. Neil, B. Reaves, and L. Williams. What are the practices for secret management in software artifacts? In *2022 IEEE Secure Development Conference (SecDev)*, pages 69–76, 2022.

[12] G. Birke. How secure are ansible vaults?, 2017. Website.

[13] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[14] M. A. D. Brogada, A. M. Sison, and R. P. Medina. Cryptanalysis on the head and tail technique for hashing passwords. In *2019 IEEE 7th Conference on Systems, Process and Control (ICSPC)*, pages 137–142, 2019.

[15] M. A. D. Brogada, A. M. Sison, and R. P. Medina. Head and tail technique for hashing passwords. In *2019 IEEE 11th International Conference on Communication Software and Networks (ICCSN)*, pages 805–810, 2019.

[16] Chick3nman. Title of the gist. `https://gist.github.com/Chick3nman/32e662a5bb63bc4f51b847bb422222fd`, 2023. GitHub Gist.

[17] G. Contributors. Github pyca cryptography. `https://github.com/pyca/cryptography`, 2023. GitHub Repository.

[18] G. Contributors. Pypi warehouse. `https://github.com/pypi/warehouse`, 2023. GitHub Repository.

[19] G. Contributors. Python docker library snapshot. `https://github.com/docker-library/python/tree/`, 2023. GitHub Repository Snapshot.

[20] P. D. Contributors. pip secure installs. `https://pip.pypa.io/en/stable/topics/secure-installs/`, 2023. Documentation Webpage.

[21] A. Documentation. Execution environments. `https://docs.ansible.com/automation-controller/latest/html/userguide/execution_environments.html`, 2023. Documentation Webpage.

[22] J. Egner. Password strength in dollars. `https://jacobegner.blogspot.com/2020/11/password-strength-in-dollars.html`, November 2020. Blog Post.

[23] I. Fjolla. cryptography-tampered. `https://github.com/ingmarfjolla/cryptography-tampered`, 2023. GitHub Repository.

[24] I. Fjolla. reproducer-material-ansiblevault. `https://github.com/ingmarfjolla/reproducer-material-ansiblevault`, 2023. GitHub Repository.

[25] B. Grewell. Cracking ansible vault secrets with hashcat. `https://www.bengrewell.com/cracking-ansible-vault-secrets-with-hashcat/`, 11/25/2023. Website.

[26] B. Hitaj, P. Gasti, G. Ateniese, and F. Perez-Cruz. Passgan: A deep learning approach for password guessing. In R. H. Deng, V. Gauthier-Umaña, M. Ochoa, and M. Yung, editors, *Applied Cryptography and Network Security*, pages 217–237, Cham, 2019. Springer International Publishing.

[27] hops. Hashcat/hashcat: Pull request #1643. `https://github.com/hashcat/hashcat/pull/1643`, 2018. GitHub Pull Request.

[28] ingmarfjolla. Add support for scrypt as a key derivation function 82104. `https://github.com/ansible/ansible/pull/82104`, 2023. GitHub Pull Request.

[29] G. Khawaja. *Cryptography and Hash Cracking*, pages 319–344. 2021.

[30] kholia. Openwall/john: Pull request #3171. `https://github.com/openwall/john/pull/3171`, 2018. GitHub Pull Request.

[31] kholia. ansible2john.py. `https://github.com/openwall/john/blob/bleeding-jumbo/run/ansible2john.py`, 2023. GitHub Repository.

[32] A. Krause, J. H. Klemmer, N. Huaman, D. Wermke, Y. Acar, and S. Fahl. Pushed by accident: A mixed-methods study on strategies of handling secret information in source code repositories. In *In 32nd USENIX Security Symposium, USENIX 2023, Anaheim CA, USA, August 9-11, 2023*. USENIX Association, Aug 2023.

[33] P. Ladisa, H. Plate, M. Martinez, and O. Barais. Sok: Taxonomy of attacks on open-source software supply chains. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1509–1526. IEEE, 2023.

[34] K. Marchetti and P. Bodily. John the ripper: An examination and analysis of the popular hash cracking algorithm. In *2022 Intermountain Engineering, Technology and Computing (IETC)*, pages 1–6, 2022.

[35] M. Meli, M. McNiece, and B. Reaves. How bad can it git? characterizing secret leakage in public github repositories. 01 2019.

[36] A. Name. Docker hub repositories hide over 1,650 malicious containers. `https://www.bleepingcomputer.com/news/security/docker-hub-repositories-hide-over-1-650-malicious-containers/`, 2023. News Article.

[37] L. Neil, S. Mittal, and A. Joshi. Mining threat intelligence about open-source projects and libraries from code repository issues and bug reports. In *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 7–12, 2018.

[38] L. Neil, H. S. Ramulu, Y. Acar, and B. Reaves. Who comes up with this stuff? interviewing authors to understand how they produce security advice. In *Nineteenth Symposium on Usable Privacy and Security (SOUPS 2023)*, pages 283–299, Anaheim, CA, Aug. 2023. USENIX Association.

[39] R. Opdebeeck, A. Zerouali, and C. De Roover. Smelly variables in ansible infrastructure code: Detection, prevalence, and lifetime. In *Proceedings of the 19th International Conference on Mining Software Repositories*, MSR '22, page 61–72, New York, NY, USA, 2022. Association for Computing Machinery.

[40] R. Opdebeeck, A. Zerouali, and C. De Roover. Control and data flow in security smell detection for infrastructure as code: Is it worth the effort? In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, pages 534–545, 2023.

[41] P. Percival. The scrypt password-based key derivation function. RFC 7914, Internet Engineering Task Force (IETF), August 2016. RFC Document.

[42] A. Rahman, F. L. Barsha, and P. Morrison. Shhh!: 12 practices for secret management in infrastructure as code. In *2021 IEEE Secure Development Conference (SecDev)*, pages 56–62, 2021.

[43] A. Rahman, R. Mahdavi-Hezaveh, and L. Williams. A systematic mapping study of infrastructure as code research. *Information and Software Technology*, 108:65–77, 2019.

[44] A. Rahman, C. Parnin, and L. Williams. The seven sins: Security smells in infrastructure as code scripts. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 164–175, 2019.

[45] A. Rahman, M. R. Rahman, C. Parnin, and L. Williams. Security smells in ansible and chef scripts: A replication study. *ACM Trans. Softw. Eng. Methodol.*, 30(1), jan 2021.

[46] N. Saavedra and J. a. F. Ferreira. Glitch: Automated polyglot security smell detection in infrastructure as code. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ASE '22, New York, NY, USA, 2023. Association for Computing Machinery.

[47] W. Scacchi, J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani. Understanding free/open source software development processes. *Software Process: Improvement and Practice*, 11:95–105, 03 2006.

[48] Setu Basak, Lorenzo Neil, Bradley Reaves, and Laurie Williams. What are the practices for secret management in software artifacts? In *Proceedings of the IEEE Secure Development Conference*.

[49] V. S. Sinha, D. Saha, P. Dhoolia, R. Padhye, and S. Mani. Detecting and mitigating secret-key leaks in source code repositories. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 396–400, 2015.

[50] snovvcrash. Pentest / infrastructure / devops / ansible. `https://ppn.snovvcrash.rocks/pentest/infrastructure/devops/ansible`, 11/25/2023. Website.

[51] R. User. Ansible vault vs supercomputer. `https://www.reddit.com/r/ansible/comments/11tgnmc/ansible_vault_vs_supercomputer/`, 11/25/2023. Reddit Post.

[52] S.-F. Wen. Software security in open source development: A systematic literature review. In *2017 21st Conference of Open Innovations Association (FRUCT)*, pages 364–373, 2017.

[53] D. Wermke, J. H. Klemmer, N. Wöhler, J. Schmüser, H. S. Ramulu, Y. Acar, and S. Fahl. "always contribute back": A qualitative study on security challenges of the open source supply chain. In *In Proceedings of the 44th IEEE Symposium on Security and Privacy (IEEE S&P '23)*. IEEE Computer Society, May 2023.

[54] A.-K. Wickert, L. Baumgärtner, F. Breitfelder, and M. Mezini. Python crypto misuses in the wild. In *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ESEM '21, New York, NY, USA, 2021. Association for Computing Machinery.