CVE Project

This project has an individual part and a group part.

Individual. For the individual project, you will investigate a password hashing CVE.

Group. For the group project, you will work in **teams of four or five** to investigate another CVE in depth. You may choose your group. Each group will submit a single solution.

The work your group submits must be entirely your group's own work. You may consult with other students about the conceptualization of the project and the meaning of the questions, but you may not look at any part of someone else's solution or collaborate with anyone outside your group. You may consult published references, provided that you appropriately cite them (e.g., with program comments), as you would in an academic paper.

All parts of the project are due via Gradescope (except for one item found in Section 1.2), following the submission checklist below. Your submission MUST include the following information:

1. List of people in your group
2. List of people you discussed the project with (outside your group)
3. List of references used (online material, course notes, textbooks, Wikipedia, etc.)

If any of this information is missing, at least 20% of the points for the assignment will automatically be deducted from your assignment. See also discussion on plagiarism and the collaboration policy on the course syllabus.

1 Introduction

The Common Vulnerabilities and Exposures (CVE) database organizes a vast collection of security vulnerabilities across many different applications. When browsing the database, you will notice that there are types of vulnerabilities that always resurface. Some of these examples include memory safety, padding/error oracles, and poor implementation/lack of password hashing.

The individual part of this project is examining a CVE involving password hashing. This includes a short report and a coding exercise where you reverse password hashes.

The group part of this project involves an open-ended investigation into a CVE of your choice. You will choose a single CVE and examine the vulnerability identified. By the end of this project, you will be able to accurately explain the attack, its impact, and the solution implemented.

The project has the following deliverables:

1.1 Individual: Password hashing CVE

1. Complete the password hashing lab (coding). See instructions in Section 2.1.
2. Write a 1-2 page report on one of the listed password hashing CVEs below. See requirements for this report in Section 2.2.
3. Both parts are due **Friday November 4, 2022** on Gradescope.

1.2 Group: Open-ended CVE investigation

1. Write a 1-2 paragraph proposal for a CVE that your group would like to investigate. This writeup will be used to determine if the selected CVE meets criteria for further investigation. See instructions in Section 3.1.

   Ping the instructors on Piazza with a link to a Google doc of your writeup, so we can check that your CVE is suitable. Please also link your Google doc on the project signup sheet. https://docs.google.com/spreadsheets/d/1M71w_zYoeaeIdMCOCy7oJuJSj7xWjfObycj2SLSUpm4/edit#gid=1750634430

   Note: This is the one item that you do not need to submit via Gradescope.

   Approval is required before the group may proceed to the next step. Complete this deliverable before the topic approval deadline. **Due: Friday October 28, 2022**.
2. Write a 5-6 page report detailing your findings. See requirements for report in Section 3.2. **Due: Friday November 11, 2022** on Gradescope.

2 Individual: Password hashing CVE

2.1 Password hashing lab

You will write python code to reverse password hashes in order to crack passwords. You will define three functions, for which some starter code will be provided. The starter code can be found on Piazza.

2.1.1 Part 1

You should write a program that finds a password given its SHA256 hash, assuming that the password is a number between 1 and 500000. The input to your program should be a list of strings,

where each string is SHA256 hash of an integer. You should output a list of integers. An example test case is:

```
hash_to_int(["6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b",
"d4735e3a265e16eee03f59718b9b5d03019c07d8b6c51f90da3a666eec13ab35"]) → [1, 2]
```

2.1.2 Part 2

You should write a program that finds a password given its salted SHA256 hash, assuming that the password is a number between 1 and 500000. **In this case, the salt is prepended to the password.** The input: a list of tuples of size 2, where the first element is a string salt and the second element is a string hash. You should output a list of integers. An example test case is:

```
salted_hahes_to_int([("salt",
"dc90cf07de907ccc64636ceddb38e552a1a0d984743b1f36a447b73877012c39"),
("salt2","6626e0d24e096d7af41b2c4d3c56335f6c451c3ef26bde0b7d4343318b3bafc2")])
→ [1, 2]
```

2.1.3 Part 3

This will be similar to part 2, except we have extended your starter code to apply a "slow" hash. This time the passwords will be hashed by composing SHA256 with itself 50 times in a row. Additionally, the salt will consist of one digit(0-9). Your job is again to find the underlying passwords. **Again, in this case, the salt is prepended to the password.** An example test case is:

```
hard_salted_hashes_to_int([("1",
"bc781d39b6449b3c7f4cd66f36987a30f93fc5747f8d85e7a4526538358b25c5"),
("1","5fb62bdcae6a6579e10e7693d57ed0f22107439307ca9b3381af608c2854310d")])
→ [1,2]
```

2.1.4 Submission and grading

For all three parts, your program will be tested against inputs other than the ones provided above. You should only edit the `lab3.py` file, no other code should be necessary. Your code should be as efficient as possible.

We will grade based on the functionality of your code, and that your code is properly documented.

2.2 Part 4: Short report

Choose a single CVE from the following list:

- CVE-2022-0022
- CVE-2022-25012
- CVE-2022-1666

Write a short 1-2 page report about your selected CVE. Address the following elements:

1. Short description of the vulnerability and the product/system affected. Why does it matter that this product has a password hashing vulnerability? Who could be affected by this vulnerability? How could their security be harmed?
2. Who can launch the attack? Detail the requirements an attacker needs to perform the attack. Think about what type of privileges/resources an attacker needs in order to exploit the password hashing vulnerability.
3. Threat model. Use diagrams and/or narrative to model an attack that exploits the password hashing vulnerability and uses it to harm a user of the affected product or system. Explain the different players (attacker, company selling the product, users, and other interacting parties).

NOTE: We are looking for you to explain the boarder context of the attack; not just that passwords can be learned, but what could an adversary do with a leaked password. (For example, in last year's Colonial Pipeline incident, a lost VPN password lead to gas shortages in the US. (See this.))

3 Group: Open-ended CVE investigation

3.1 Selecting a CVE

You should select a CVE from 2022 (i.e. CVEs that are numbered with 2022-XXXX) that is related to cryptography or web security - though there are CVEs related to other aspects of security, they are outside the scope of this project. The following website allows you to search for CVEs and filter for recency:

https://nvd.nist.gov/vuln/search

The writeup should include:

- a list of your group's members
- the selected CVE. Include identification number, link to CVE database entry, and other related links.
- a *brief* (1-2 paragraph) description of the attack.

You should choose a CVE where you have some rudimentary understanding of the underlying vulnerability BEFORE beginning your research. It should be clear in your 1-2 paragraph writeup that you know the basics of the concepts touched upon by the vulnerability.

Please do not pick one of the listed CVEs in Section 2.2 or one of the CVEs from the example projects provided by the instructors.

3.2 Full Report

Write a 5-6 page report about your approved CVE. Your report should address the following elements:

1. Short description/introduction.
2. Who can launch the attack? Detail the requirements an attacker needs to perform the attack.
3. Why does the attack matter? Which industries/systems are impacted?
4. What damage does the CVE cause? (crash bug vs. RCE vs. privacy leak)
5. What is the attack? How does it work? Explain in rigorous details.
6. How was the bug fixed? Explain whether the patch is a real fix or just a stopgap that could be circumvented later.
7. Threat model. Use diagrams and/or narrative to model the entire attack. Explain the different players (attacker, company, users, and other interacting parties).
8. Vector justification. Each CVE is assigned a score determining its impact. This score is calculated by a set of categories: Attack vector, attack complexity, privileges required, user interaction, scope, and confidentiality/integrity/availability impact. Enumerate through these categories and write for each one a short paragraph arguing whether you agree/disagree with the assigned categorization of your CVE. See Section 3.3 for a list of all the different vector categories.

See the sample report uploaded on Piazza as an example for what we're looking for.

3.3 CVE vector categories

Find your CVE's vector listed in the "Severity" section of the NIST NVD database entry.

3.3.1 Attack vector (AV)

Network (AV:N): A vulnerability exploitable with Network access means the vulnerable component is bound to the network stack and the attacker's path is through OSI layer 3 (the network layer). Such a vulnerability is often termed "remotely exploitable" and can be thought of as an attack being exploitable one or more network hops away (e.g. across layer 3 boundaries from routers).

Adjacent network (AV:A): A vulnerability exploitable with Adjacent Network access means the vulnerable component is bound to the network stack, however the attack is limited to the same shared physical (e.g. Bluetooth, IEEE 802.11), or logical (e.g. local IP subnet) network, and cannot be performed across an OSI layer 3 boundary (e.g. a router).

Local (AV:L): A vulnerability exploitable with Local access means that the vulnerable component is not bound to the network stack, and the attacker's path is via read/write/execute capabilities. In

some cases, the attacker may be logged in locally in order to exploit the vulnerability, or may rely on User Interaction to execute a malicious file.

Physical (AV:P): A vulnerability exploitable with Physical access requires the attacker to physically touch or manipulate the vulnerable component, such as attaching an peripheral device to a system.

3.3.2 Attack complexity (AC)

Low (AC:L): Specialized access conditions or extenuating circumstances do not exist. An attacker can expect repeatable success against the vulnerable component.

High (AC:H): A successful attack depends on conditions beyond the attacker's control. That is, a successful attack cannot be accomplished at will, but requires the attacker to invest in some measurable amount of effort in preparation or execution against the vulnerable component before a successful attack can be expected.

3.3.3 Privileges required (PR)

None (PR:N): The attacker is unauthorized prior to attack, and therefore does not require any access to settings or files to carry out an attack.

Low (PR:L): The attacker is authorized with (i.e. requires) privileges that provide basic user capabilities that could normally affect only settings and files owned by a user. Alternatively, an attacker with Low privileges may have the ability to cause an impact only to non-sensitive resources.

High (PR:H): The attacker is authorized with (i.e. requires) privileges that provide significant (e.g. administrative) control over the vulnerable component that could affect component-wide settings and files.

3.3.4 User interaction (UI)

None (UI:N): The vulnerable system can be exploited without interaction from any user.

Required (UI:R): Successful exploitation of this vulnerability requires a user to take some action before the vulnerability can be exploited, such as convincing a user to click a link in an email.

3.3.5 Scope (S)

Unchanged (S:U): An exploited vulnerability can only affect resources managed by the same authority. In this case the vulnerable component and the impacted component are the same.

Changed (S:C): An exploited vulnerability can affect resources beyond the authorization privileges intended by the vulnerable component. In this case the vulnerable component and the impacted component are different.

3.3.6 Confidentiality impact (C)

None (C:N): There is no loss of confidentiality within the impacted component.

Low (C:L): There is some loss of confidentiality. Access to some restricted information is obtained, but the attacker does not have control over what information is obtained, or the amount or kind of loss is constrained. The information disclosure does not cause a direct, serious loss to the impacted component.

High (C:H): There is total loss of confidentiality, resulting in all resources within the impacted component being divulged to the attacker. Alternatively, access to only some restricted information is obtained, but the disclosed information presents a direct, serious impact.

3.3.7 Integrity impact (I)

None (I:N): There is no loss of integrity within the impacted component.

Low (I:L): Modification of data is possible, but the attacker does not have control over the consequence of a modification, or the amount of modification is constrained. The data modification does not have a direct, serious impact on the impacted component.

High (I:H): There is a total loss of integrity, or a complete loss of protection. For example, the attacker is able to modify any/all files protected by the impacted component. Alternatively, only some files can be modified, but malicious modification would present a direct, serious consequence to the impacted component.

3.3.8 Availability impact (A)

None (A:N): There is no impact to availability within the impacted component.

Low (A:L): There is reduced performance or interruptions in resource availability. Even if repeated exploitation of the vulnerability is possible, the attacker does not have the ability to completely deny service to legitimate users. The resources in the impacted component are either partially available all of the time, or fully available only some of the time, but overall there is no direct, serious consequence to the impacted component.

High (A:H): There is total loss of availability, resulting in the attacker being able to fully deny access to resources in the impacted component; this loss is either sustained (while the attacker continues to deliver the attack) or persistent (the condition persists even after the attack has completed). Alternatively, the attacker has the ability to deny some availability, but the loss of availability presents a direct, serious consequence to the impacted component (e.g., the attacker cannot disrupt existing connections, but can prevent new connections; the attacker can repeatedly exploit a vulnerability that, in each instance of a successful attack, leaks a only small amount of memory, but after repeated exploitation causes a service to become completely unavailable).