habr.com /ru/articles/771688/

Вас сдаст Гитхаб: деанонимизация пользователей SSH-серверов

Deleted user ⋮⋮ 03.11.2023

Автор оригинала: Mark Pashmfouroush

Недавно в своих ежедневных чтениях я наткнулся на явление, о котором думал уже много лет: феномен утечки информации людей, использующих SSH. Этот тип утечки информации не является новым явлением. Я давно предупреждал об этой опасности своих друзей, использующих SSH, но мой голос услышали лишь несколько человек. Поэтому я решил дать пояснения по этому поводу, потому что я считаю, что необходимо понимать этот риск в ИТ-сообществе, особенно в нынешней ситуации. Я буду исходить из предположения, что у вас, дорогой читатель, есть опыт работы с SSH.

Асимметричные ключи SSH

Протокол SSH использует асимметричную криптографию. Короче говоря, для шифрования связи и клиент и сервер каждый имеют открытый ключ и закрытый ключ, при установлении соединения они обмениваются друг с другом только открытым ключом, чтобы информация каждой стороны шифровалась с его помощью, а после обмена данные будут расшифрованы с помощью закрытого ключа. SSH использует тот же метод и для так называемой "авторизации по ключам". Существует распространенное мнение, что этот метод более безопасен, чем аутентификация по паролю, и это, безусловно, правда.

Чтобы подключиться, клиент должен сначала отправить открытый ключ на хост на другой стороне, которым обычно является SSH-сервер или Git.

Github, Gitlab и открытые ключи

В предыдущем абзаце я говорил о подключении к серверам Git через SSH. GitHub — один из нескольких известных сервисов Git, у которого много пользователей из нашей страны. GitHub позволяет пользователям работать с Git, используя публичные ключи SSH, но проблема в том, что такие сервера, как GitHub, публикуют ключи своих пользователей... в открытом виде. Вы можете увидеть один аспект этого увлекательного, но вредного действия GitHub, перейдя по ссылке ниже и подставив вместо "username" свое имя пользователя:

```
https://github.com/username.keys
```

Злоумышленники или работники силовых органов, имеющие какие-то намерения против своих граждан, могут просто просканировать эти SSH-ключи и создать базу данных аккаунтов и ключей, а дальше устанавливать совпадения с открытыми ключами, обнаруженными в других местах, и

использовать их для идентификации граждан. Например, вы можете проверить наличие вашего ключа SSH на GitHub с помощью команды (`ssh git@github.com`), и если такой ключ существует, вы увидите свое имя пользователя GitHub в ответе сервера.

Короче говоря, если у вас действительно есть SSH-ключ, зарегистрированный в GitHub, вам нужно принять за данность, что все ваши SSH-ключи GitHub вполне возможно уже просканированы и хранятся в какой-то базе данных.

Опасность сопоставления открытых ключей

Чтобы понять, как утечка открытого ключа пользователей SSH ставит под угрозу конфиденциальность, рассмотрим следующий эксперимент.

У меня есть два ключа SSH и сервер, который принимает от меня только один ключ. Теперь я пытаюсь подключиться к серверу с обоими ключами, а затем сравниваю логи клиента (с флагом verbose). Обратите внимание, что я удалил некоторые строки вывода из-за большого количества логов.

```
$ ssh -v -o "IdentitiesOnly=yes" -i ~/.ssh/id_ed25519 root@10.2.10.5
OpenSSH_9.1p1, OpenSSL 3.0.7 1 Nov 2022
debug1: Connecting to 10.2.10.5 [10.2.10.5] port 22.
debug1: Connection established.
debug1: Authenticating to 10.2.10.5:22 as 'root'
debug1: Host '10.2.10.5' is known and matches the ED25519 host key.
debug1: Will attempt key: /home/mark/.ssh/id_ed25519
debug1: Authentications that can continue: publickey,password,keyboard-
interactive
debug1: Next authentication method: publickey
debug1: Offering public key: /home/mark/.ssh/id_ed25519
debug1: Server accepts key: /home/mark/.ssh/id_ed25519
Authenticated to 10.2.10.5 ([10.2.10.5]:22) using "publickey".
```

```
$ ssh -v -o "IdentitiesOnly=yes" -i ~/.ssh/id_rsa root@10.2.10.5
OpenSSH_9.1p1, OpenSSL 3.0.7 1 Nov 2022
debug1: Connecting to 10.2.10.5 [10.2.10.5] port 22.
debug1: Connection established.
debug1: Authenticating to 10.2.10.5:22 as 'root'
debug1: Host '10.2.10.5' is known and matches the ED25519 host key.
debug1: Will attempt key: /home/mark/.ssh/id_rsa
debug1: Authentications that can continue: publickey,password,keyboard-
interactive
debug1: Next authentication method: publickey
debug1: Offering public key: /home/mark/.ssh/id_rsa
debug1: Authentications that can continue: publickey,password,keyboard-
interactive
```

```
debug1: Next authentication method: keyboard-interactive
(root@10.2.10.5) Password:
```

Приведенный выше пример показывает, что в первом случае клиент предлагает серверу свой открытый ключ, сервер его принимает. Но во втором логе публичный ключ не принимается. Но вопрос в том, реально ли с помощью одного только публичного ключа, но не имея приватного, определить, разрешено пользователю, обладающим этим ключу подключение к серверу или нет? Обязательно ли для такой проверки иметь и приватный ключ? Кажется, что да, но давайте проверим доказательство этой гипотезы.

Для проверки я написал короткий код и попробовал использовать открытый ключ на целевом сервере, без использования приватного.

```
packagemain



import (
"fmt"
"io"



"golang.org/x/crypto/ssh"
)

const (
username="root"
server="10.2.10.5:22"
publicKey="ssh-ed25519
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
)



funcmain() {
parsed, _, _, _, err :=ssh.ParseAuthorizedKey([]byte(publicKey))
iferr!=nil {
panic(err)
        }



signer :=&DummySigner{PubKey: parsed}
authMethod := []ssh.AuthMethod{ssh.PublicKeysCallback(
func() ([]ssh.Signer, error) {
return []ssh.Signer{signer}, nil
```

```
                },
        )}


config :=&ssh.ClientConfig{
User:            username,
Auth:            authMethod,
HostKeyCallback: ssh.InsecureIgnoreHostKey(),
        }


_, _=ssh.Dial("tcp", server, config)


ifsigner.Accepted {
fmt.Println("Public key was accepted by server")
return
        }


fmt.Println("Public key was rejected by server")
}

typeDummySignerstruct {
PubKeyssh.PublicKey
Acceptedbool
}

                func (signer *DummySigner) PublicKey() ssh.PublicKey {
        return signer.PubKey
}
func (signer *DummySigner) Sign(rand io.Reader, data []byte)
(*ssh.Signature, error) {
        signer.Accepted = true
        return &ssh.Signature{Format: signer.PubKey.Type()}, nil
}
```

Результат:

```
Public key was accepted by server
```

Как и ожидалось, все сработало.

Что в итоге?

Мы увидели, что с помощью нескольких строк кода можно протестировать, разрешен ли человеку с определенным публичным SSH-ключом (даже не имея его приватного ключа) доступ на сервер. Соответственно, тестирование того или иного ключа на большом количестве IP-адресов не должно стать проблемой. С помощью этого метода, используя данные из открытых источников, можно легко получить связки "юзернеймы - открытые ключи", а потом идентифицировать сервера, которые этим юзерам принадлежат.

Речь идет не о взломе сервера или учетной записи GitHub, а о сборе общедоступной информации о людях и организациях. Поэтому имейте в виду, что раз ваш публичный ключ светится в открытом доступе где-нибудь на Github, и вы установили этот же ключ для пользователя root или угадываемого имени пользователя для входа в систему на вашем сервере, то сопоставить эти два факта будет довольно легко. Я рекомендую использовать два разных публичных ключа для действий, связанных с Git, и для вашего сервера, и не публиковать нигде даже публичные ключи, используемые на сервере.

Использованная литература:

SSH WHOAMI.FILIPPO.IO

whoarethey: Determine Who Can Log In to an SSH Server

От переводчика: Это статья - очередное напоминание о том, что публичные ключи - именно *публичные*, и поэтому они могут использоваться для деанонимизации пользователя, если тот использует один и тот же ключ на разных серверах.

Например, зная юзернейм интересующего вас человека на Гитхабе и получив его публичный ключ оттуда, можно сканированием IP-адресов найти сервера в сети, которые он администрирует.

Либо другой случай, имея базу "юзернейм - публичный ключ", дампнутую с Гитхаба, теоретически можно найти аккаунт человека, который администрирует интересующий вас сервер - перебирать придется очень долго, но все-таки реально (особенно если отфильтровать набор юзеров для перебора по каким-то дополнительным критериям). А при возможности MitM-перехвата трафика (помните недавнюю историю с Hetzner и Jabber.ru?) где-то по пути до сервер, подслушав публичный ключ узнать аккаунт его администратора на Гитхабе можно вообще одной командой.

Совет все тот же: использовать разные публичные ключи для разных целей.