

# Logging

---

James Brucker

# KBank data stolen

Pipit Aneaknithi, Kasikornbank president, revealed that on July 25, KBank found that 3,000 names of corporate customers using KBank's website for the letter of guarantee service might have been leaked.

As soon as KBank detected the irregularity, it said it immediately closed the loophole... The data that may have been leaked was the names and telephone numbers of KBank's corporate customers using the letter of guarantee service via the website only.

The Nation August 01, 2018 18:47

<http://www.nationmultimedia.com/detail/business/30351237>

# KTB customer data stolen

---

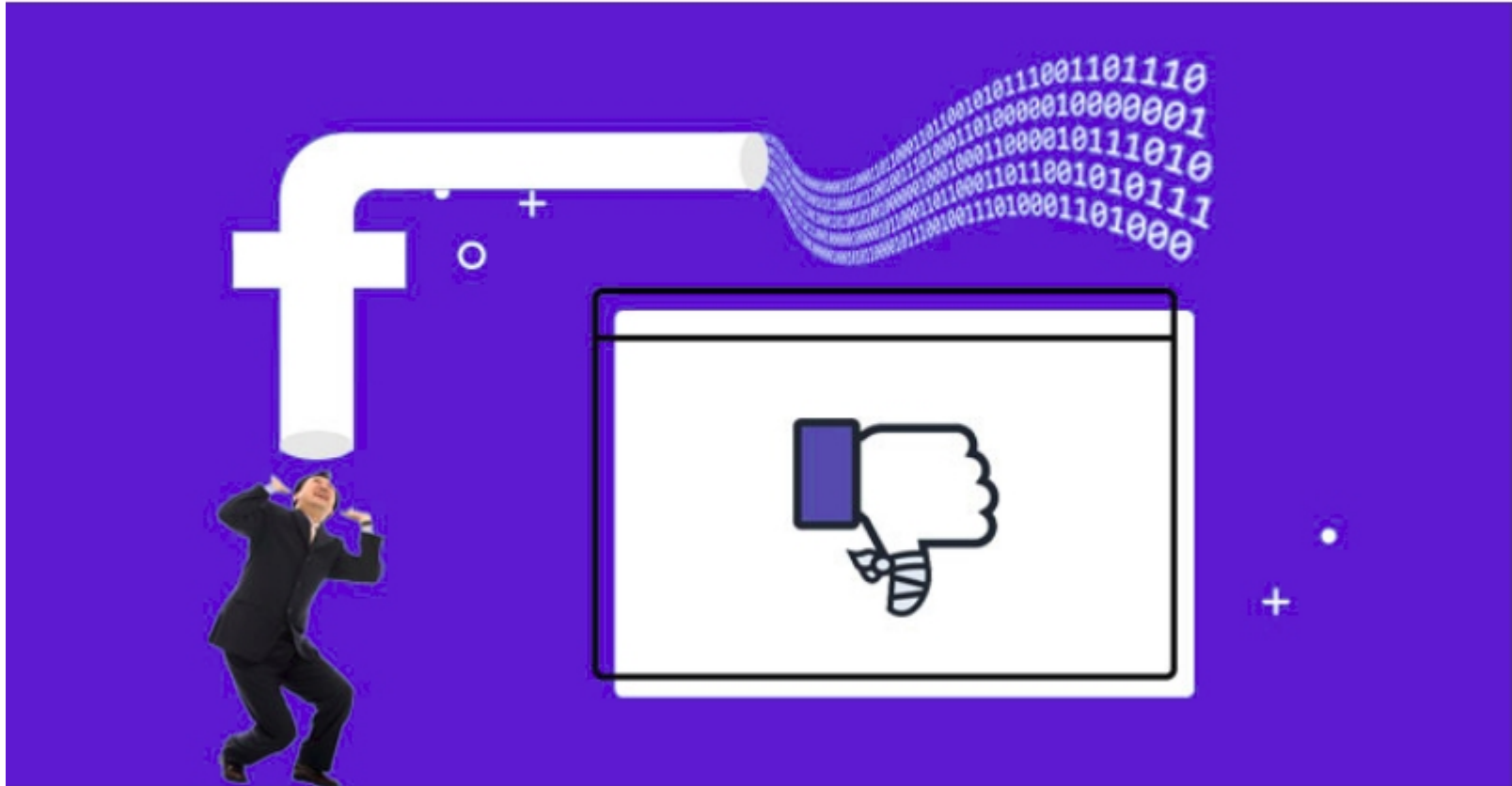
KTB president Payong Srivanich said that the bank had detected **general information** from **120,000 retail customers** who applied for mortgages and personal loans online ... **was hacked** in the days leading up the July holidays.

Bangkok Post - August 01, 2018 04:00

<https://www.bangkokpost.com/news/security/1513410/kbank-ktb-targeted-in-cyber-attacks>

# 30 Million Facebook Accounts Were Hacked: Cl... Them

📅 October 12, 2018    👤 Swati Khandelwal



Late last month Facebook announced its [worst-ever security breach](#) that allowed an unknown group of hackers to steal secret access tokens for millions of accounts by taking advantage of a flaw in the 'View As' feature.

# How Facebook was Hacked

---

A flaw in "View As" feature that enables someone to preview a page as another user. Flaw has been present since 2017, but first **detected** on 14 Sep 2018 due to **rise in suspicious activity**.

Facebook knows [exactly which accounts were hacked](#) and you can check your account.

<https://www.wired.com/story/how-facebook-hackers-compromised-30-million-accounts/>

<https://thehackernews.com/2018/10/hack-facebook-account.html>

# How Did They *Know*?

---

**How** did **KBank** **know** 3,000 customer's data stolen ...  
and what data was stolen?

**How** did **KTB** **know** 120,000 customers "who applied for  
a mortgage or loan" had data stolen?

**How** did **Facebook** **know** whose data was stolen?

# LOGGING

---

They keep "logs" of events and activity.

# Linux Logs Almost Everything

---

Unix/Linux keep logs for many services in `/var/log`.

Typical logs are:

`auth.log` - authentication related (login, sudo)

`boot.log` - system start-up (boot) activity

`dpkg.log` - package install and configuration messages

`kern.log` - messages from the kernel

`lastlog` - most recent login by each user

`ufw.log` - firewall messages

Log files are automatically **rotated** every 1 - 7 days, so they do not become too large.



# Python Logging

---

`logging` - Python logging package

`# get a named logger. Use a module or app name`

```
logger = logging.getLogger( "test" )
```

`# log messages at different log levels`

```
logger.debug( "a message for developers." )
```

```
logger.info( "this is a boring info message" )
```

```
logger.warning( "a warning you should pay attention to" )
```

```
logger.error( "An error occurred." )
```

```
logger.critical( "A critical error or failure" )
```

```
logger.log( logging.INFO, "another INFO level mesg" )
```

# 5 Log Levels

---

## Level Names:

CRITICAL = 50

ERROR = 40

WARNING = 30

INFO = 20

DEBUG = 10

Any log level

FATAL = 50

## Example:

```
logger.critical("Can't connect to db")
```

```
logger.error("Error rendering template")
```

```
logger.warning("Failed login by ...")
```

```
logger.info("Successful login by ...")
```

```
logger.debug(request)
```

```
logger.log( level, message )
```

an *alias* for CRITICAL

# What to Log?

# log some events

```
logger.critical( "Connection to database failed" )  
logger.error( "Poll question has no choices: "+question )  
logger.warning( "Failed login by " + form.username )  
logger.info( "Successful login by " + user.username )  
logger.debug( f"foo(x) called with x = {x}" )
```

# log an exception

try:

```
    q = Question.objects.get(id=1)
```

except Exception as **ex**:

```
    logger.exception("Expected question not found", ex)
```

# Where to Log?

---

Use a configuration file to specify where log messages are stored.

- Console, *aka* Standard Output
- a File
- Database
- Network - send to a network log server

You can use more than one destination, or route log messages based on log level or source.

# You Can Control Logging

---

## 1. Turn logging on, off, or filter:

*"Only print messages of level WARNING or higher"*

```
logging.setLevel( logging.WARNING )
```

*"... but for 'polls' loggers, print INFO messages too"*

```
logging.getLogger('polls').setLevel( logging.INFO )
```

## 2. Redirect the log messages to a file or other service

## 3. Change the format of log messages

# Example log configuration in code

```
# Default:  only print WARNING or higher
```

```
logger = logging.getLogger( )
```

```
logger.info( "This message is not printed" )
```

```
logger.warn("This is a warning")
```

```
This is a warning
```

```
# Set message threshold level to INFO (or higher)
```

```
logger.setLevel( logging.INFO )
```

```
logger.info( "This message IS printed" )
```

```
This message IS printed
```

```
logger.debug( "this is not printed" )
```

# Logging Practice

---

Instructions: Logging practice on course github.io site

`demo_log.py` code you can use (also on github.io site)

# Loggers Form a Named Hierarchy

---

Root logger:            `root = logging.getLogger( )`  
'polls' logger:        `log1 = logging.getLogger('polls')`  
'polls.models' log:   `log2 = logging.getLogger('polls.models')`

## Loggers Inherit from Parent (Ancestor) loggers

Configure the root logger.

Other loggers get those settings by default.



# Log Message Propagation

If `logger.propagate = True` (the default) then events logged to this logger will be passed to handlers of higher level loggers, in addition to handlers of this logger. Threshold & filters of ancestor loggers are ignored.

[see "class logging.Logger" in Python Docs]

```
root = logging.getLogger( )      - the root logger  
loga = logging.getLogger( 'a' )  - descendent of root  
logb = logging.getLogger('a.b') - descendent of 'a' & root
```

```
logb.warn("Warning!") - sent to logb, then loga, then root
```

*You must **try this yourself** in order to understand what it does.*

# Use module name for logger name

---

You can use any name you want for logger.

Standard is to use **module name** or **package name**

Using a good name helps you **track source of messages**.

## 1. A logger named for this module

```
log = logging.getLogger( __name__ )
```

## 2. A logger for "polls" app

```
log = logging.getLogger( "polls" )
```

# getLogger( "name" ) is a Singleton

Only one instance for each named logger.

*Good! We can get a logger whenever we need it.*

```
log1 = logging.getLogger( "auth" )
```

```
log2 = logging.getLogger( "auth" )
```

```
log3 = logging.getLogger( "AUTH" )
```

```
# are they the same object?
```

```
log1 is log2
```

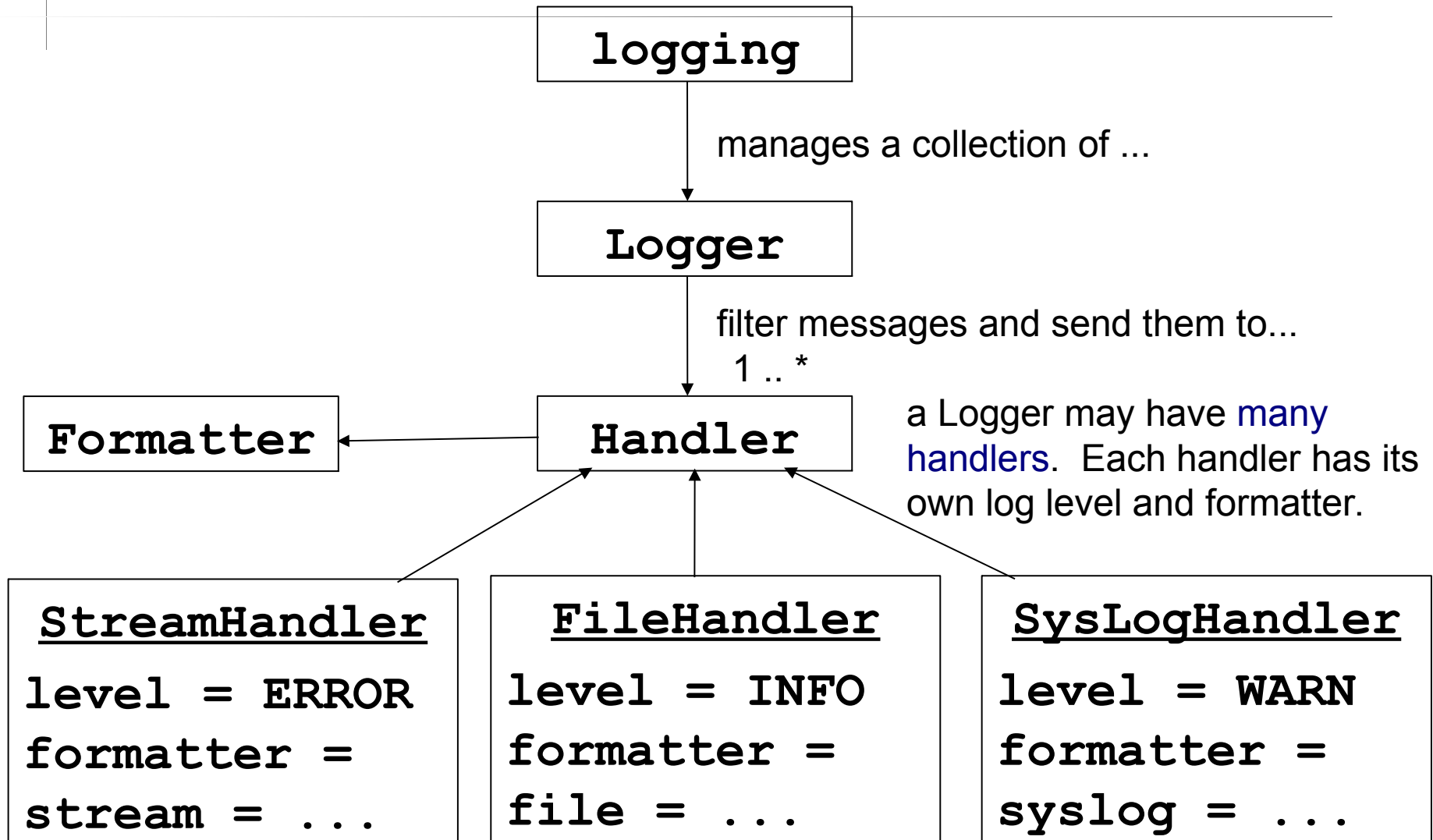
```
True
```

```
# is the logger name case sensitive?
```

```
log1 is log3
```

```
False
```

# Logging Architecture



# Configuration Example

**# A handler that writes to a file**

```
filehandler = logging.FileHandler( "/tmp/demo.log" )
```

**# This handler should log everything**

**# Note that logger's own log-level may override this.**

```
filehandler.setLevel( logging.DEBUG )
```

**# Message format is: 2019-10-28 10:45:23 a.b INFO: hi there**

```
formatter = logging.Formatter(  
    "%(asctime)s %(name)s %(levelname)s: %(message)s"  
)
```

**# Tell file handler to use this formatter**

```
filehandler.setFormatter( formatter )
```

**# Add it to root logger**

```
root = logging.getLogger( )
```

```
root.addHandler( filehandler )
```

# Separation of Responsibilities

---

Draw a UML class diagram showing relationship between:

Logger

Handler

FileHandler

ConsoleHandler

Formatter

One we didn't cover: `Filter`

# Why Separate Responsibilities?

---

1. What is the benefit of separating Logger, Handler, and Formatter?

Imagine if we had:

StreamLogger

FileLogger

RotatingFileLogger

SyslogLogger

...

2. Is there a *Design Principles* that recommends this design?

FYI: Log4J & SLF4J use the same design.

# Why Separate Responsibilities?

---

## *Design Principles*

### Single Responsibility Principle

Don't Repeat Yourself - use delegation & Strategy Pattern

- avoid duplicate code, duplicate logic, duplicate bugs

Open-Closed Principle - we can *extend* functionality of Logging by writing our own Handler or Formatter.



# How to Use the 5 Log Levels

---

What *should* you log to each of these levels?

CRITICAL

ERROR

WARNING

INFO

DEBUG

See: *Python Logging Tutorial*

<https://docs.python.org/3/howto/logging.html>

# What do the method names tell you?

```
# Tell file handler to use this formatter
```

```
filehandler.setFormatter( formatter )
```

```
# Add handler to root logger
```

```
logger = logging.getLogger( )
```

```
logger.addHandler( filehandler )
```

Why is one named "setSomething" and the other "addSomething"?

# Log Handlers and Formatters

---

Python has many Log Handlers you can choose:

<https://docs.python.org/3/library/logging.handlers.html>

## **Important Handlers:**

`logging.StreamHandler(stream=sys.stdout)`

`logging.FileHandler( filename )`

`logging.RotatingFileHandler( filename, maxBytes=... )`

`logging.TimeRotatingFileHandler`

`logging.SysLogHandler(address=("localhost",port),...)`

# Web App Logging

---

Web Apps have some special concerns:

1. want to know IP address for events and activity
2. Web app may be deployed on many hosts, and may not be persistent. How can you make separate logs from web app?
3. How to aggregate logs from different parts of app?

# Web App Logging

---

*What events or activity should a web app log?*

1. Login - username, IP address, date-time
2. Logout
3. Errors and exceptions
4. Deployment
5. User activity - at least all activity that changes something
6. Invalid requests

# Logging Done Wrong

---

Facebook stored 200 - 600 Million users' passwords in **plain text** in **log files** for years.

<https://krebsonsecurity.com/2019/03/facebook-stored-hundreds-of-millions-of-user-passwords-in-plain-text-for-years/>



# What Info Should You Log When...

---

1. A failed or successful login occurs.

- > username

- > IP address

- > date/time

2. A user submits a "vote" to the polls application.

- > question and choice he voted

- > which session or IP address he voted from

- > date/time

# Learn Python Logging

---

*Logging HOWTO* in the Python Library docs

- Use guide in the Advanced Tutorial.
- Don't use the Basic Tutorial (static log methods)
- <https://docs.python.org/3/howto/logging.html>

*Logging - Logging Facility for Python*

- in the Python library docs
- configuration, using formats, and handlers

**How to Configure Logging**

- <https://docs.python-guide.org/writing/logging/>
- **3 ways:** .INI file, a dict or JSON file, function calls



# Learn Python Logging

---

## Logging Cookbook

<https://docs.python.org/3/howto/logging-cookbook.html>

# Django Logging

---

Django uses Python Logging, adds some "conveniences".

See: Django User Guide, section on Logging (only 10 pages with many examples)

Configuration: Django uses JSON-format text to configure loggers in `settings.py`.

# Configure Django Logging

---

```
LOGGING = {
    'disable_existing_loggers': False,
    'handlers': {
        'file': {
            'level': 'DEBUG',
            'class': 'logging.FileHandler',
            'filename': '/path/to/myapp.log',
        },
        'console': {
            'class': 'logging.StreamHandler'
        }
    },
    'loggers': {
        'myapp': {
            'handlers': ['console'],
            'level': 'INFO',
            'propagate': False,
            ...
        }
    }
}
```

# Logging Advice

---

1. Configure the root logger, but don't use it directly.
2. For deployed web apps, log to console (12FactorApp)
3. Configure logger via config variables, not settings.py.
  - OK to partially configure in settings.py but get details from configuration file

# Java Has More Log Levels

java.util.logging

SEVERE

WARNING

INFO

( CONFIG )

FINE - stupid name

FINER - stupider

FINEST

OFF

ALL

Log4J & SLF4J Levels

FATAL

ERROR

WARN

INFO

DEBUG

TRACE

OFF

ALL



# Experience at KU

---

# Chinese are Attacking My Server!

---

In `/var/log/auth.log` on `se.cpe.ku.ac.th`:

Nov 18 06:29:48 se sshd[6720]: Failed password for root  
from 116.31.116.16 port 61430 ssh2

Nov 18 06:29:52 se sshd[6720]: message repeated 2  
times: [ Failed password for root from 116.31.116.16  
port 61430 ssh2]

Someone is trying to login as root.

Where is 116.31.116.16?

Search Google...

# 116.31.116.16

---

**116.31.116.16 | ChinaNet Guangdong Province Network | AbuseIPDB**

<https://www.abuseipdb.com/check/116.31.116.16>

116.31.116.16 has been reported 409 times. ... 116.31.116.16 was first reported on December 3rd 2017 , and the most recent report was 4 hours ago .

**IP List of Brute force attackers**

<https://report.cs.rutgers.edu/DROP/attackers>

... 115.186.147.235 115.249.205.29 116.196.76.135 116.31.116.11 116.31.116.12  
116.31.116.14 116.31.116.16 116.31.116.21 116.31.116.23 116.31.116.24 ...

**The Anti Hacker Alliance™ fights against 116.31.116.20**

<https://anti-hacker-alliance.com/index.php?ip=116.31.116.20>

116.31.116.x



# The "Fix"

---

1. ssh was already configured to deny root login.
2. add firewall rule to deny all traffic from IPs in China.