

АЯКАКАПИ

Bimonthly Cyber Security Magazine

04

March - April

www.arkakapimag.com



The Future of Security and Reliability

Secret Sharing Systems

Database Attacks and Protection Methods

Malware Programming and Lockdown Virus

Network Programming with Scapy

ISSN 2645-906X



9 772645 906009

Hello world!

I would like to start by thanking all of you, dear readers for your support and love for our magazine, and the staff who work non-stop to bring this magazine to light. Behind the scenes, there is a lot going on to deliver what we believe is beneficial to the world. Even if there is one person who could see the beauty in the cybersecurity and informatics world through this magazine, we could not be any happier!

On March 8, we will be celebrating International Women's Day. Yet, the origin of the date makes us appreciate what we have today and commemorate what some women did to make us live this way. The origin of the day is March 8, 1857, where some garment workers in New York City protested against the working conditions and low wages. The protests continued, i.e., on March 8, 1908, women marched again in New York City for voting rights, ending child labor and better wages.

As of this significant date, we would like to remind you of some women who rewrote history and without whom the tech world - simply the *world itself* - would never have been the way it is now. The long list starts with Ada Lovelace and continues with Hedy Lamarr,

Grace Hopper, and Top Secret Rosies, and so on...

In the last issue's Editor's Note, we talked a bit about the Universal Declaration of Human Rights - *UDHR*. Supporting, respecting and maintaining women rights means respecting the humankind and our future: ourselves, as the UDHR applies to everyone and each and every woman has rights as any other person on this planet does. The value of a person does not depend on their gender, religion, language, skin color, age, etc., but only on the things they do.

As of this issue, you can find interesting articles on encryption, database attacks, viruses, network programming and much more! To learn is to protect yourself. Read so you can learn, learn so you can protect yourself.

Special thanks to Netsparker Ltd. for sponsoring this issue!

Enjoy!

Cansu TOPUKÇU
editor@arkakapimag.com

ARKAKAPI MAG

Cyber Security Magazine YEAR: 1 – MARC-APR ISSUE: 2 Bimonthly - ISSN: 2645-906X www.arkakapimag.com

Editor in Chief: Ziyahan Albeniz • ziyahan@arkakapimag.com

Editorial Operations Manager: Cansu Topukçu • cansu@arkakapimag.com

Chief Business Officer: Oğuz Aydınılmaz • oguz@arkakapimag.com

Publishing Coordinator: Şahin Solmaz • sahin@arkakapimag.com

Director of Web: Ömer Çıtak • omer@arkakapimag.com

Legal Advisor: Mehmet Pehlivan • mehmet@arkakapimag.com

Assistant research editor: Ayşenur Burak • nurayse47@gmail.com

Translators: Hakan Özer, Zekvan Arslan, Atalay Keleştemur, Nuri Çilengir, Enes Özen

Social Media: twitter.com/arkakapimag instagram.com/arkakapimag facebook.com/arkakapimag

We are proud to secure all our emails with Tutanota.

CONTENT

CYBER SECURITY CONFERENCES - Ayşenur Burak	4
THE FUTURE OF SECURITY AND RELIABILITY - Chris Stephenson	6
CRYPTOLOGY FRONT OF WORLD WAR II THE ENIGMA ENCRYPTION MACHINE - Bayram Gök	10
SECRET SHARING SYSTEMS - Ceren İnce	18
DATABASE ATTACKS AND PROTECTION METHODS - Ömer Faruk Colakoğlu	22
MALWARE PROGRAMMING AND LOCKDOWN VIRUS - Bener Kaya	46
NETWORK PROGRAMMING WITH SCAPY - Güray Yıldırım	51
SIBER YILDIZ 2019 WRITEUP - Esra Nur Soylu	57
A SECURITY GUIDE FOR YOUR ANDROID DEVICE - Arka Kapı	78
SIGNAL INTELLIGENCE SIGNAL LISTENING AND ANALYSIS METHODS - Murat Şişman	84

netsparker

Web Application Security Scanner

Use Netsparker to Identify Exploitable Vulnerabilities and Other Security Flaws in Your Websites, Web Applications & Web Services Before Hackers Do.

Netsparker scanners employ the unique, dead accurate & fast **Proof-Based Vulnerability Scanning Technology** that automatically verifies the identified vulnerabilities with a proof of exploit, so you do not have to manually verify them.



Trusted by

 ERNST & YOUNG
Quality in Everything We Do

 NASA

 SAMSUNG



 ISACA
The International Information Systems Audit and Control Association

 Microsoft

 ING

 Booz | Allen | Hamilton

 SIEMENS

Cyber Security Conferences

INTERNATIONAL CONFERENCE ON BIG DATA & DATA SCIENCE

March 04-05, 2019

**Hotel Augusta Barcelona Valles
Barcelona, Spain**

EuroSciCon is organizing the 8th Edition of International Conference on Big Data & Data Science 2019 which will be held between March 04-05 at Barcelona.



Info: <https://big-data.euroscicon.com/>

DEVOPSDAYS LOS ANGELES

March 08, 2019

**Pasadena Convention
Center, United States**

This conference is dedicated to the DevOps community, and professionals.

Info: <https://www.devopsdays.org/events/2019-los-angeles/welcome/>



CLOUD & CYBER SECURITY EXPO 2019

March 12-13, 2019

ExCeL London

It is the only place that gives you everything you need to learn, wherever you are in your digital transformation journey and to stay safe in an increasingly hostile digital environment. It's quite simply the industry-leading event for digital-age guardianship.

Info: <https://www.cloudsecurityexpo.com/>

FIRST CYBER THREAT INTELLIGENCE SYMPOSIUM 2019

March 18-20, 2019

BT Centre London, United Kingdom

There will be one day of training followed by two days of plenary sessions. This event will be open to both FIRST members and non-members.

Info: <https://first.org/events/symposium/london2019/>



CYBER SECURITY FOR ENERGY & UTILITIES MASTERCLASS 2019

April 09-12, 2019

Singapore, Malaysia

It will provide power, electricity, energy & utilities company attendees with the tools and know-hows of how to plan and strategically develop a cyber security strategy.

Info: <http://www.equip-global.com/cyber-security-for-energy-amp-utilities-2019>

HARTFORD CYBERSECURITY CONFERENCE

April 18, 2019

Hartford, United States

Data Connectors is host the Cybersecurity Strategies Conference.

Includes Keynote Session and CISO Panel

Info: <https://dataconnectors.com/events/hartford2019/>



IOT TECH EXPO GLOBAL

April 25-26, 2019

Olympia London, United Kingdom

The World's Largest IoT Conference Series; the IoT Tech Expo Global event in London will bring together key industries from across the globe for 2 days of top level content and discussion. Exploring the latest innovations within the Internet of Things and covering the impact it has on many industries including Manufacturing, Transport, Supply Chain, Insurance, Logistics, Government, Energy and Automotive, this conference is not to be missed.

Info: <https://www.iottechexpo.com/global/>



The Future of Security and Reliability

This article is a call. On behalf of security, the question of “patch or treatment?” needs to be answered.

We’re dealing with patches for now. For the future, this will not be enough!

The size and complexity of operating systems and software are continually increasing; the Linux kernel contains more than 20 million lines of code, and that is just the core. The operating system is a whole lot more than that.

Your equipment is equally sophisticated. There are chips with up to 20 billion (you read it right; it is billions, not millions) transistors; 10 billion transistor processors are now becoming standard.

In both areas, we have systems designed for ancient needs. C language thus Unix is 45, Microsoft Windows is 32, Linux is 27, Intel x86 architecture is 40, and ARM 33 years old.

Therefore, it is challenging to consolidate these systems, which had not been designed due respect to safety or reliability.

With the word security, we want to express the immunity of a system against possible attacks. With reliability, we mean how much a system can do without stopping the work it promises.

Considering that the computer manages aircraft, cars, medical devices that keep us alive, and more, both security and reliability are crucial.

In the future, IoT (Internet of Things) will increase our dependence on reliability and security of software.

Past and Present Solutions

Among the software history, the software that has the oldest history of high security have been those of airplanes and spacecraft. In the past, they tried to solve this problem by augmenting the number of computers. Votes were taken for critical operation between several computers that owned different hardware and separately written software. Space Shuttle had five control computers in two different types. Generally, numerous computer systems are used in the field of Avionics (aircraft electronics). Even though it saves the day up to a point, it had never been the ultimate solution. In addition to multiple systems, it is necessary to perform tests that are systematic and to try to predict every possible situation as in mathematics.

Here, unfortunately, you must pause for some readers. In some countries, the *proof* subject is not in the high school syllabus.

For example, I have the following theorem: “This computer system does not allow the aircraft to stall” - an important theorem. Preconditions, axioms, assumptions, are all crucial, of course. However, the theorem just existing is way safer than flying the plane a few times and saying “Look, it’s not down!”. However, the methods usually used for software are more like the second one.

Nevertheless, in addition to systematic tests, formal methods, i.e., methods requiring proof, have been used for these critical systems.

For example, if you look at the Airbus 330/340 software guide, we see how formal methods are being used.

30.13 Development Environment

The development of each Airbus Industrie aircraft has been supported by an Iron Bird whole-aircraft systems rig, and by supporting systems rigs that enable work to proceed simultaneously, without mutual interference. The A330/A340 model is no exception, and a number of facilities have been constructed specifically for this programme. These methods are now being used by other airframe manufacturers.

Proper software development is an essential part of systems development throughout the aircraft, and a number of software tools have been developed, notably in the areas of formal methods, rapid prototyping, automatic coding, and rapid data recovery and analysis. These are supplemented by large, fast data recording and telemetry facilities on the test aircraft fleet, associated with real-time and rapid-playback test data displays for the benefit of the flight test observers on board the test aircraft and for the test and systems engineers on the ground.

The result of this environment, the proper use of features from previous programmes, and the proper management of test data flow and the resulting decision process, has created an aircraft that has had a remarkable trouble-free period of introduction into service. This is true both in terms of customer satisfaction and in terms of measurable parameters such as delay rate, which have been up to an order of magnitude better for A340 than for the previous derivative long-range aircraft that entered service.

The Avionics Handbook

Another field that is currently using formal methods is the robot spacecraft that go to Mars.

In an article by a NASA Department, the JPL (Jet Propulsion Laboratory), it describes how the practical application of the software in the vehicle by formal methods can be practiced by various mathematical techniques. In any case, a vehicle that is 100 million kilometers away should be prevented from entering a vicious circle.

Commercial software is both expensive and not practical because of the size of the software with formal methods.

Solved in the Future

In the fifth edition of Arka Kapi Dergi, I mentioned the seL4 kernel. Such initiatives represent the future of safety and reliability.

It was a nice coincidence that a scientific article about a system using the seL4 kernel was published in CACM magazine after my article was published.

Boeing's AH-6 autonomy, the unmanned helicopter is not a small drone or UAV; it is a full-size helicopter. This system has always been tested for its reliability, but a red team hacked the helicopter system successfully in 2013.

The attackers managed to direct the helicopter to any location or to crash the helicopter via a USB device attached to the helicopter. The original helicopter system is based on Linux. Linux could not be completely eliminated. However, a virtual machine was created under the seL4 kernel to isolate Linux and the non-critical software, for instance, the camera control of the helicopter, was kept there. This software was blocked from accessing other software flying the helicopter.

Besides, a White Box attack has been attempted - White Box attacks are the attacks in which the attacking team is provided with all the source code and documentation of the system as well as root access to the weaker camera system. Despite all this, the red team could not affect the helicopter's flight systems.

So how was this accomplished? First, a proven operating system, seL4 microsppheres, was used. The core itself, the C compiler, and other software are systems proven with formal methods. The proofs have been tested on the software under the virtual machines isolated one by one under the core. Since they were not isolated from each other, the same criteria were not applied to each software.

By implementing such methods, the project implementers could retrofit an existing system and make the system more secure.



Boeing Little Bird flying - did you notice the pilot's absence?

Of course, this is possible under certain conditions: the seL4 microsphere is not a 20 billion line like a Linux kernel, but rather 10-thousand-line C code. Such an attempt for Linux is impossible.

The Future of Proofs

How are security and reliability ensured outside space and aircraft? This cannot be done with software consisting of millions of lines and languages such as C. Even if there are no other factors, the “undefined behavior” trap of the C language alone is enough to make it difficult. In many cases, programs written in C may exhibit different behaviors. This is a disaster for security. In practice, to avoid the poor results of this quality of the C language, both programs, and compiler behavior should be severely restricted. It would be enough to read John Regehr’s articles to understand the severity of this problem. While there are 200 different undefined behaviors in C, it is very difficult to prove the behavior of pure C programs.

Formal methods are now becoming more widely accepted. As an example of highly critical problems, we can mention the programs running on browsers. Typically, these programs are written in Javascript. Unfortunately, Javascript is not challenging about variable types, is also complicated, and does not have a clear

formal system to recognize the meaning of programs. That’s why WebAssembly has been developed for client programs with the joint venture of Google, Apple, Mozilla, and Microsoft. Unlike other languages, the primary feature of WebAssembly is its formal form, mathematical semantics (definitions that define the meaning of programs in language) and type system. We will explain the importance of the types in a moment.

Proof and type systems will be important. New languages and new operating systems are yet to come. New concepts such as proof, type system, formal semantics in new languages will be important. We have to get used to and learn from them.

The importance of types

If we desire security and reliability, we have to say goodbye to our 40-year-old friends, Unix, and C language. Instead, we should adopt provable micro-cores like seL4, as an operating system. However, what will be our solution for the programming languages?

It is not a language that makes it easy for us to write good programs. What we need is a language that makes it impossible for us to write the wrong programs. It’s a target. For theoretical reasons, we may

not reach this goal fully at 100%. However, we still have to protect this goal. Adding some important attributes to our languages will make it easier for us to use proofs in larger programs.

In programs, there has to be referential transparency, i.e., when a function is evaluated with the same parameters, it must be guaranteed to produce the same result. Therefore, in our language, mutation (changing the value of the variable) should be impossible.

In our software language, behaviors that may cause undefined behavior should be banned, i.e., rejected by the compiler. This requires a powerful type system.

To realize these, we need a functional language. Nowadays, the languages we already have at hand may not be enough; they are complex and robust too much. Some undesired uncertainties come with Turing Completeness.

However, it is no coincidence that the seL4 operating system reference application was written in Haskell; Haskell is a *pure functional typed* language.

The necessity of types can be explained with two simple examples. We have two major security vulnerabilities. One is Buffer Overflow, and the other is SQL injection. Both of them are due to type mismatch.

If a user enters a text and an SQL command is not of the same type, our compiler may block SQL injection attacks without any further measure. So it becomes impossible to write a program that might be exposed to SQL injection. Buffer Overflow is a more difficult issue. Of course, in environments such as Java language, it is possible to prevent vulnerabilities - such as Buffer Overflow - during runtime, but what our main issue is to prevent it in the compilation stage.

We have to solve these problems so that we can trust the software which occupies more and more space each day in our everyday lives. There will be compromises just like every engineering problem. Nevertheless, proven software will gain more importance. Our old operating systems and programming languages will not be enough.

Conclusion

As usual, I would strongly recommend that you read the original articles that I refer to as references.

Learn programming languages with powerful type systems such as Haskell and Rust. Learn about proof systems such as Coq. If you want to work with security and reliability, this is the future.

References:

1. <https://www.linuxcounter.net/statistics/kernel>
2. EPYC: A Study in Energy Efficient CPU Design Nathan Brookwood
<https://www.amd.com/system/files/documents/The-Energy-Efficient-AMD-EPYC-Design.pdf>
3. Architecture of the space shuttle primary avionics software system Gene
- D. Carlow Communications of the ACM CACM Volume 27 Issue 9, Sept. 1984 pp 926-936
4. New Avionics Systems —Airbus A330/A340 J. P. Potocki de Montalk
http://www.davi.ws/avionics/TheAvionicsHandbook_Cap_30.pdf
5. Exploiting Traces in Static Program Analysis, Alex Groce, Rajeev Joshi <https://agroce.github.io/sttt08.pdf>
6. Formally Verified Software in the Real World, Gerwin Klein, June Andronick, Matthew Fernandez, Ihor Kuz, Toby Murray, Gernot Heiser, Communications of the ACM Volume 61 Issue 10, October 2018
7. Mathematically Verified Software Kernels: Raising the Bar for High Assurance Implementations Dr. Daniel Potts, Rene Bourquin, Leslie
- Andresen, Dr. June Andronick, Dr. Gerwin Klein, Prof Gernot Heiser
8. Meltdown, Spectre ve Foreshadow, Yaklaşan Devrimin Ayak
- Sesleri, Chris Stephenson, Arka Kapı Dergi Sayı 4
9. A Guide to Undefined Behavior in C and C++, John Regehr, <https://blog.regehr.org/archives/213>
10. Bringing the web up to speed with WebAssembly Andreas Rossberg, Ben L. Titzer, Andreas Haas, Derek L. Schuff, Dan Gohman, Luke Wagner, Alon Zakai, J. F. Bastien, Michael Holman Communications of the ACM Volume 61 Issue 12, December 2018 Pages 107-115

Cryptology Front of World War II

The Enigma Encryption Machine

It is one of the most famous machines humans have invented. It's the unforgettable star of cryptology. Its story has been the subject of several films from Hollywood. We all know it as the Enigma ciphering device used by the Nazi army during the Second World War. Also, many of the cryptographic devices that developed after enigma have inspired by it. Enigma is the cause of many young people who are interested in cryptology.

When the Second World War was over, the German army had about a hundred thousand Enigma devices in its hands. It is now possible to see the Enigma device, where all the intelligence units have sacrificed their lives to capture one, and the famous cryptanalysts spent years trying to solve them, in museums. You can also find many sites sell replicas of enigma machine on the internet. Speaking of buying, look carefully to the typewriters you see at junk dealers. One of them can be an Enigma device worth 45 thousand euros.

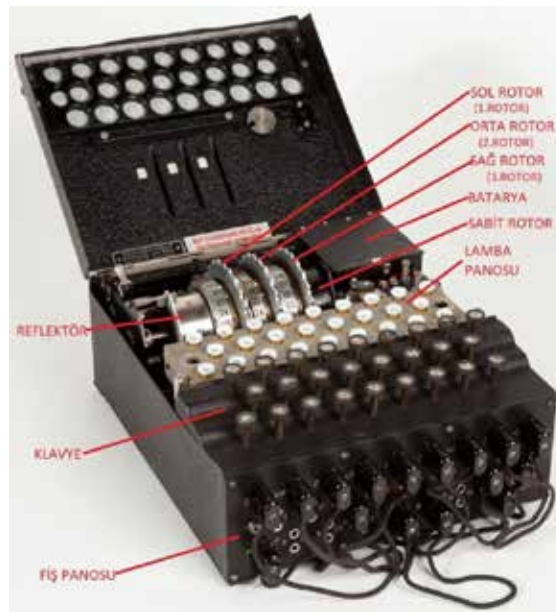


Image 1 : Enigma Device with Three Rotor

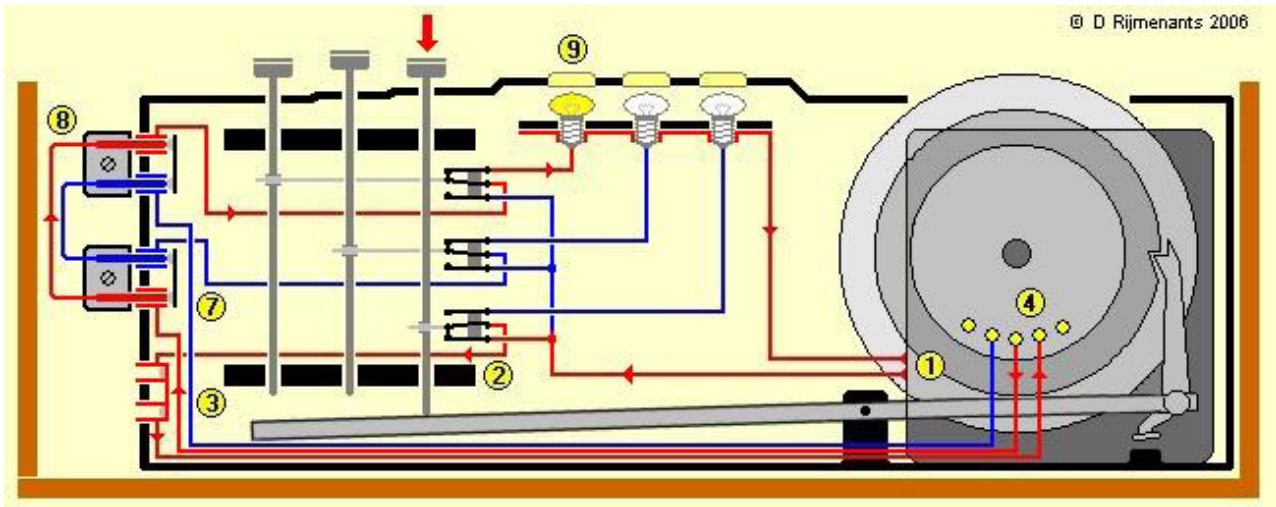
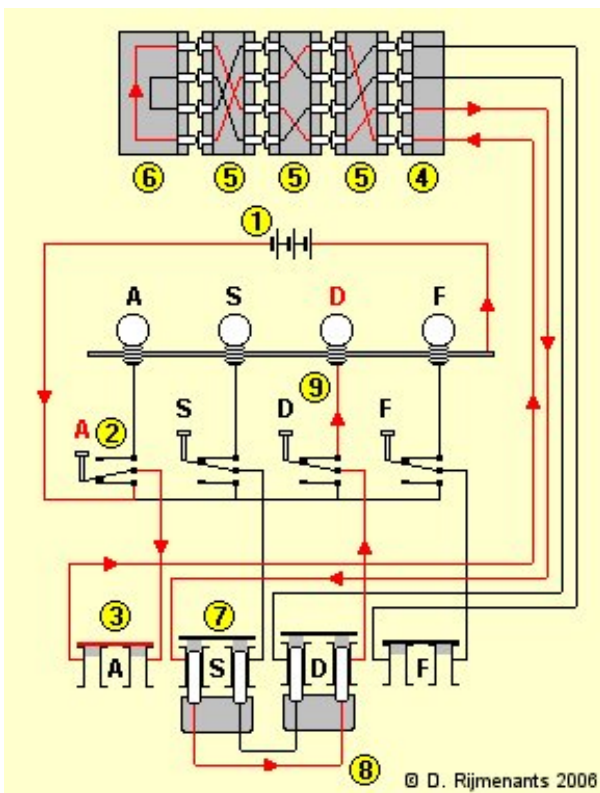


Image 2: Electro-mechanical scheme of Enigma Device [1]



1. Battery
2. Switch electric key (A pressed)
3. Plugboard (No patch cables)
4. Stationary Rotor
5. Rotor
6. Reflector
7. Plugboard (Patch cables present, S<->D)
8. Patch cable
9. Lamp panel (D light on)

Open Text= A

Ciphered Text= D

Image 3: Electro-mechanical scheme of Enigma Device [2]

History

The Enigma machine was patented and manufactured by Dr. Arthur Scherbius on February 23, 1918. The first model weighed about 50 kg and was quite bulky. Marketed with the Enigma brand, which means “Riddle” in Greek, the device was developed to be used as a typewriter in its wooden box and in the form of portable 12-pound mobile military models.

As a device designed for commercial purposes, the Enigma did not show the desired sales success until the German Army showed interest. In 1926, the German Navy (Reichsmarine) began using the Funkschlüssel C model, which was specially adapted for them. After some improvements, The Enigma G model, designed for the German Army (Reichswehr) in 1928, was widely used by all German military units and other government agencies after 1930 as Enigma I.

How Does Enigma Work?

To help us understand, we can imagine that the Alberti disk is an advanced continuation of Caesar cipher, the Vigenere cipher is the advanced continuation of Alberti disk, and The Enigma cipher is the advanced continuation of the Vigenere cipher. Each of them has been developed to address the lack of the previous method. While the Caesar cipher was satisfied with a single key (letter shift), multiple keys (letter shift) could be used on the Alberti disk. With the help of a selected keyword in Vigenere cipher, it was possible to encrypt each letter using a different key (different characters shift). In each method, the key mechanism was somewhat complicated. Enigma is the most perfect and last of this series.

The most obvious difference of Enigma is that when each letter is encrypted, it automatically changes the keyword to be used for the next letter to be encrypted. In practice, the method allowed the use of key number close to infinite. It is therefore almost impossible to find repeated strings attached to the same key, such as the Vigenere cipher, in encrypted text. The first key must be known or found to decrypt it.

Enigma is an electromechanical device that works with a battery. They all work on the same basic principle and chassis, although it contains mechanical differences according to the model.

Let's look at the parts of the device.

Keyboard

The enigma is equipped with a 26-letter QWERTY keyboard. They probably wanted it to be compatible with the Morse code. Each key is associated with both a switch and a mechanical pedal that allows the rotor to move step by step. The electrical switch is connected directly to the letter input on the plugboard and to the lamp panel. It does not have a function in the encryption process, it is for data entry purposes.

Plugboard

Is the section where the encryption process begins. This model has a letter change board that provides additional security that is not available in previous models. There are plug entries representing each of the 26 letters on the board. It allows swapping a pair of letters both on the keyboard and on the lamp panel. As can be seen in Figure 4, the letters A and M are replaced by a patch cord attached between the letters A and M. When we press A we will get M and vice versa, so A becomes M and M becomes A, and we get the letter itself if the letter change cable is not attached. A = A; M = M. The number of joining according to the number of patch cables to be used is given in Table 1. The German Army usually used 10 patch cables. This means about 48 bits. The air force was using the UHR (clock) plug attached to the plugboard. With the help of a rotatable key on the UHR plug-in, they were able to easily change the pre-set combinations.

Cable(n)	Combinations
0	1
1	325
2	44.850
3	3.453.450
4	164.038.875
5	5.019.589.575
6	100.391.791.500
7	1.305.093.290.000
8	10.767.019.640.000
9	53.835.098.190.000
10 *	150.738.274.900.000
11	205.552.193.100.000
12	102.776.096.500.000
13	7.905.853.580.550
Toplam	532.985.208.200.000

Table 1: Number of plugboard patch cord

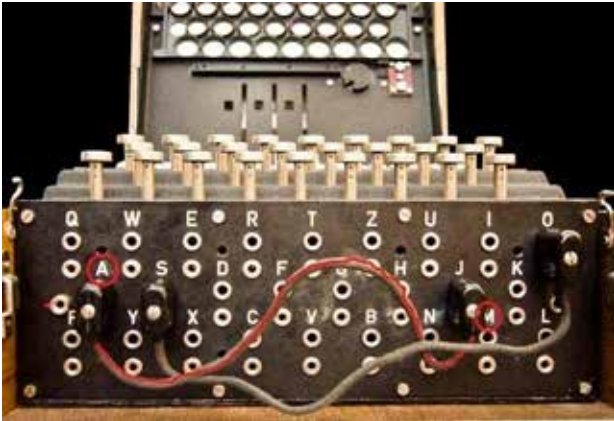


Table 2: Plugboard

Stationary Rotor

The static rotor has conductive surfaces that allow the cables coming from the plugboard to contact the pins on the moving rotor. It has no effect on the encryption process.



Image 5: Black Round Track is Stationary Rotor, silver colored 3 Piece is Latches

Rotor

The heart of the Enigma device, the most complex part, can be easily removed, replaced or mounted on an axis side by side in different order according to the encryption key. There are pins on the right side of the rotor that come in contact with the conductor surfaces of the previous rotor, and pins on the left side that come in contact with the conductor wires that transmit the signal to the next rotor through the cross-connected wires. The German army units have differentiated the cross-links and have used various rotors numbered by Roman numerals.

The German Army and Air Force used the enigma device with three rotors numbered I, II and III. In 1938, the number of rotors was increased to 5 by adding rotors IV and V. During encryption, 3 of these 5 rotors were selected and used. Selecting 3 of 5 rotors gives $5 \times 4 \times 3 = 60$ combinations.

The Germans were trying to surround the British with their submarines as they did in World War I. Special attention was therefore paid to the communication of the naval forces. In February 1942, the Enigma machine was specially modified for naval forces and coded as M4. The reflector was made thinner and the 4th rotor, which was thinned could be mounted between the reflector and the left-most rotor. There are two types of additional rotor called beta and gamma. The fourth rotor was not moving automatically, unlike the other three rotors, but was manually fixed to one of the 26 positions. The naval forces initially used 6 rotors in the same spaces as the three-rotor version. Later, rotors VII and VIII were added.

All rotors have 26 special notches on the right side and 1 special notch on the left side. When the rotor is mounted on the axis, the right side of the rotor is attached to the left side of the neighbor rotor. As seen in Image 5, there are three latches that center the joining line of the rotor, with a nail on the tip, depending on the key mechanics. When pressing any key, three latches rise upwards simultaneously. With the rising movement, the tip of the right-hand latch is attached to one of the 26 pins to the right of the rotor and moves the rotor one step further. During the nascent movement, if the tip of the center latch is aligned with one of the 26 notches to the right of the central rotor and the single notch on the left of the rotor on the right; it falls into the slot formed by two notches and locks the two rotors. The two locked rotor moves one step together. Since there is only one notch to the left of the rotor on the right, this deadlock occurs once after a lap rotation after every 26 steps of the rotor on the right. In simple term, in order to rotate the rotor in the middle, the rotor on the right should rotate 26 turns. There is also the same mechanical relationship between the third latch, the middle rotor, and the left rotor. In the M4 model, there is no latch for the fourth rotor and it's fixed because it is not connected to the mechanism. In order for the rotor to return to the start position, it is necessary to press a key $26 \times 26 \times 26 = 17576$ times. This is 17 thousand 576 combinations. Since the step-by-step movement changes the contact conductor pins and surfaces,

the encryption key also changes continuously. If we had pressed letter “A” 17.576 times in succession, we might have encountered a series of repetitive letters. You can watch the video of the mechanism through the [link](#).

Rotors VI, VII, and VIII used in the navy have 2 notches on the left side, unlike others. While the normal rotors advance the rotor on the left side of each turn, because of the 2 notches they have the rotors VI, VII and VIII rotate the rotor on the left side of the rotor by two-step for each turn.

In the first rotor produced, the letter ring and conductive pin/surface were fixed. Later, to make the encryption process more complex, the letter rings have been made rotatable through the rotor. And after rotate the ring to the needed position, it was fixed with a small lock. This simple change allowed pin/surface connections and letter relating to variable rotors. Since there is no rotor to the left of the leftmost rotor, only the ring setting of the middle and right rotor will affect the encryption. Because each ring can be set to 26 different positions, 26x26 gives 676 combination.

Pin/surface connection while Ring adjustment is at position **A** :

ABCDEFGHIJKLMN**O**PQRSTUVWXYZ
EKMFLGDQVZNTOWYHXUSPAIBRCJ

If the ring is set to position **B**, the pin/surface connection be as follows :

ZABCDEFGHIJKLMN**O**PQRSTUVWXYZ
EKMFLGDQVZNTOWYHXUSPAIBRCJ



Image 6: Side surfaces of the rotors, pins, and notches



Image 7: Internal connections of rotors, pins, and surfaces cross-linking cables, letter ring



Image 8: Letter ring lock (<http://www.cryptomuseum.com/crypto/enigma/working.htm>)

Reflector

The reflector passes through the rotors and sends the encrypted message back to the last rotor over the cross-linked return pins. The encrypted signal reaching the reflector is sent back to the last rotor from the cross-connected pin. The signal is re-encrypted by re-adding all the rotors. The customized reflector types were produced for Army intelligence (Abwehr) and the Marine Corps. The cross-linked pins of the reflector are active in the encryption process.



Image 9: Reflector

Lamp Panel

The lamp panel is an output device that reports the encrypted letter to the operator by lighting a lamp. The operator sends the encrypted message via telegraph or radio as Morse code or by courier in writing. It is connected to the plug board, such as a keyboard. Changing the letters made in the plug panel is also valid in the Lamp panel.



Image 10: Lamp Panel

To summarize, Enigma is a highly complex device. Improvements to reinforce the device later made the device more complicated. If we calculate the number of combinations given by the plug panel, rotor, ring setting, we will get $150.738.274.900.000 \times 17576 \times 676 = 1.643.9946.58.58.345.893.248$.

Usage

Enigma is a bidirectional device. Let's explain: For example, when the Enigma device is set and the letter T is pressed, it will appear that the G lamp is lit. This means that the letter T is encoded as a letter G. Again when the letter G is pressed in the same setting, the letter T will light up on the lamp panel. In other words, the Enigma device is capable of encrypting text as well as decrypting the encrypted text. There is no need for a separate device for decoding.

Each night, operators set the parameters of the enigma device according to a pre-distributed code booklet, and this setting would be valid for a full day. The rotor set for that day is selected, the letter ring is set and placed in the slot with the specified sequence. Again according to the code booklet, letter replacement settings are made. Then the message Key was created. Since the Nazis thought that tens of thousands of messages to be sent through the day would provide the statistical data needed to break the password, they designed a procedure whereby each operator could determine his or her own message key. The procedure for creating the message key implemented until 1940 was as follows.

Encryption operator

After setting the enigma device according to the code booklet:

- a) The operator randomly chose three letters. For example, **RNF** was called the main setting (Grundstellung).
- b) Then the operator would rotate the rotor manually from left to right, turning it to the **RNF** position.
- c) The operator would choose another random three letters. For example, **JRM**, it was called the message key.
- d) Then the operator presses the letters **JRMJRM** respectively and from the lamp panel, for example, notes the letters **BKTRFQ**. The **BKT** letters were called encrypted messages.
- e) Then the operator sets the rotors back to **JRM** letters and begins to encrypt the message.
- f) When the message was encrypted, some additional information (header) was transmitted to the opposite side in a specific format.

Decrypting operator

The operator on the other side also set the rotors, letter rings, and plugboards according to the parameters of the day and according to the code booklet. They use additional information (header) to resolve the message received and, respectively, follows the following procedure.

After setting the enigma device according to the code booklet:

- a) Sets the rotors to **RNF** (Grundstellung), which arrives with the message.

- b) Again, they enter **BKTRFQ**, the encrypted message setting found in the message attachment information.
- c) Obtained the **JRMJRM** message key from the lamp panel.
- d) The rotors would set the message key **JRM**.
- e) Enters the encrypted message and obtains the text that has been decoded from the lamp panel.



Image 11: One page from the codebook

In fact, the M1, M2, M3 and M4 codes given to Enigma devices are said to be referred to the encryption procedures rather than the technical specifications of the devices.

How was the legend of Enigma defeated?

There were many historical bad memories of the Polish people to be vigilant. Poland has always been crushed by the eternal competition between Russia and the leading countries of Europe. The only woman with two Nobel Prizes, Marie Curie and the Musician Chopin, were the two famous Polish people who had to leave their country because of the occupation. For this reason, the Polish people have always had good news service and skillful password breakers. Germany was followed by pure attention after the Second World War. They were the first to solve the Enigma. Commercial Enigma was already known. The Poles and the British had already decoded it. However, it is said that the Germans ‘ military Enigma shocked the Polish. They gathered their best mathematicians and established the Biuro Szyfrow, a password cracking base near Warsaw. Until 1940, the Germans changed encryption procedures several times. Biuro Szyfrow managed to break down every procedure until Poland was occupied. Both developments helped them a lot. First, the former German police, Hans-Thilo Schmidt, stole the Enigma user manual, key lists, and operating instructions from the German military cipher center, where he found work through his brother and sold it to the French secret

service. But the French failed to break the enigma. The French gave the information to Biuro Szyfrow. The second development was when the German government sent a diplomatic Enigma device to the embassy in Warsaw with an ordinary cargo. The Poles took this opportunity. They opened the package and examined the enigma device for two days and took pictures. Then they delivered the package to the embassy as if it had never been opened. The Germans didn’t realize anything. The Poles even made them two enigma devices. It is also reported that the poles have developed these devices by purchasing commercial enigma. On September 15, 1938, the Germans changed the encryption procedure once again. Marian Rejewski, who worked at Biuro Szyfrow in October 1938, and his friends developed the first cryptoanalysis device in history to decode Enigma. They called it a cryptographic bomb. As explained in Enigma usage, the encrypted message setting was transmitted in BKT in the header section. To avoid errors in the procedure used until May 1, 1940, the message key JRM was encrypted as JRMJRM twice in a row and transmitted as BKTRFQ. The first 6 letters to get the JRMJRM decoder operator were entered BKTRFQ. This procedure, which was originally designed as security, was itself vulnerable. Key transmission problems will lead to the development of open key understanding in the future. Since all documents were destroyed during the occupation, there is no detailed information about how this gap was broken. However, we have passed the Enigma settings for a full day. This procedure, which was originally designed as a security check was a security breach. Key transmission problems will enable the development of open key understanding in the future. Since all documents were destroyed during the occupation, there is no detailed information about how this vulnerability was broken. However, the enigma settings are valid for a full day.

If sufficient message keys were collected, it was possible to access the message key without the receipt board and rotor information. You can read detailed information from this [link](#). Marian Rajewski also took advantage of the initial positions chosen by lazy operators such as “AAA”, ”BBB”, and”CCC”. The number I, II, III rotors produced six combinations. They produced a cryptographic bomb for each combination. They quickly learned the internal structure of the IV and V rotors that were added about a month after the device was completed. However, five rotors produced 60 combinations. They needed more cryptological bombs. Yet they had no way of making so many cryptological bombs.

For about 7 years, The Poles decoded Enigma codes without telling the French and the British until the start of the Second World War, listening to German communication. They thought they should now share their information with the British, and handed over all the information, including two copies of The Enigma device to the British. Biuro Szyfrow destroyed all documents and cryptographic bombs before the Germans invaded Poland. The Germans never realized that the Enigma was resolved.

The British, who took over the work of solving Enigma codes from the poles and some Biuro Szyfrow workers, set up a base in Bletchley near London. They specifically targeted M4 to get rid of the German U-boats ' blockade. They gathered the best mathematicians, chess masters, puzzle enthusiasts in the country. This 8000-strong team was directly linked to Churchill, The Prime Minister of the period. In Bletchley, privacy was so important that the source of the information obtained from the Decrypted Enigma codes was said to be a spy with the code name "Boniface". Gordon Welchman and especially Alan Turing came forward from inside this team. Turing was inspired by the Polish cryptological bomb and managed to develop a cryptographic device itself. The British also called the device "bomb" to honor the

Poles. Weighing about 1 ton, this device simulates dozens of Enigma devices at the same time. 2 cryptological bombs produced by British Tabulating Machine Factory started working in March 1940. They produced 200 of these.

Alan Turing followed a different path than Marian Rejewski and his friends to break the Enigma code. Alan Turing applied "known Open Text attack" to Enigma passwords that changed every day with an automatic machine and succeeded. Frequently used words and codes such as "immediately", and "Hitler", etc. were scanned for in the ciphered text. In other words, if there is one of these words in the text, the machine understood that the message was resolved. The Turing bomb, equipped with Welchman's diagonal board, then eliminated the possibilities and accelerated the process, began to decode messages in less time. The British sought help from the Americans to break the M4, equipped with four rotors in 1942. The Americans developed their own Bombe's.

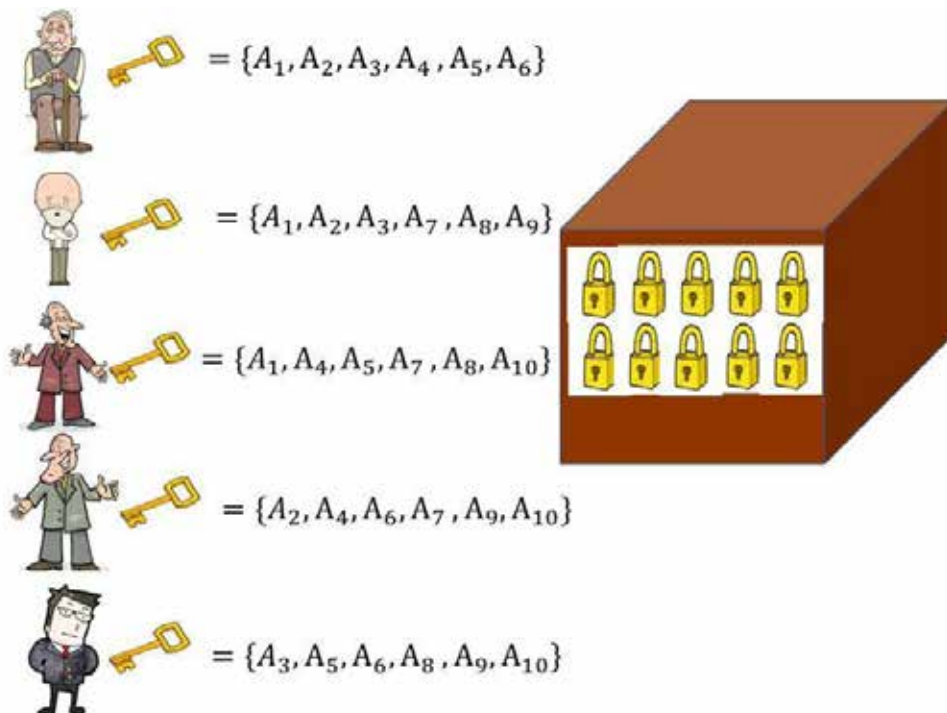
In Enigma's thrilling adventure which was invented by talented inventor Dr. Arthur Scherbius. It is our duty to discuss which of the Marian Rejewski and Alan Turing approaches is more artistic. Hail to all three geniuses!

SECRET SHARING SYSTEMS

Data storage systems have become an indispensable part of today's technology. Some companies such as YouTube, Google, and Amazon often need data storage systems that require large scale data processing. The way large-scale data is stored is probably to threaten Institution/Organization and even country security. It's very important to develop methods to secure the storage of this type of data that may threaten the institutions or organizations.

A common method of data security and encryption systems, the secret sharing systems are simply the case where the "secret" data is shared by more than one person. In other words, the secret is known only when a predetermined number of people gather together. In these systems, a distributor designs the system and determines which part of each user in the system will receive the secret.

The need for systems that don't trust a single authority has caused the development of secret sharing systems. We can think of a situation where at least 7 people from a group of 10 should give their authorization for the operation of a nuclear weapon.



Liu's problem caused the secret sharing systems to emerge in 1968. The problem is as follows: "11 scientists work on a secret project in which information is kept in a secure box. How many locks should we use to prevent the box from opening without at least 6 or more scientists? At this requirement, at least how many keys should each scientist have?"

For the solution of the problem; Without at least 6 scientists, the box should be locked with 462 locks so that the box can't be opened, and each scientist must have at least 252 keys.

The solution to this problem can be found simply with combinatorial calculations but for the most effective solutions, the problem was solved by Shamir and Blakely independently of each other in 1979. Shamir solved this problem using the interpolation formula.

Let's make an example of this problem with smaller numbers and try to understand how Shamir's method works and how effective it is by observing it again through the same example.

The Problem is as follows: 5 scientists are working on a secret project where information is kept in a safe box. How many locks should the box have to prevent the box from opening without at least 3 or more scientists? At this requirement, at least how many keys should each scientist have?

The solution can be easily solved by combinatorial methods. Without at least 3 scientists the box should be locked with 10 locks so that the box can't be opened, and each scientist must have at least 6 keys.

In the image, 10 keys are distributed so that each of them has 6 keys. When the keys of any three scientists are combined, the A_1, \dots, A_{10} keys will be obtained and the box will be opened.

Let's examine Shamir's Secret Sharing System to understand the effective solution that Shamir provides. This system based on Lagrange interpolation, which was developed to distribute a switch with certain shares within a group, is shown as follows.

The system architecture is consist of a D distributor that builds the secret sharing system, S key set, and $P=\{P_1, P_2, \dots, P_n\}$ users set. The D distributor selects s key from the S set to split it up and distribute the parts to the users according to the algorithm.

Construction Of Shamir

- Let number p be a prime number larger than the number of people n.
- Select the key to corresponding to the $a_0 \pmod p$ coefficient as a secret.
- Let random $a_1, a_2, \dots, a_{k-1} \pmod p$ coefficients be chosen to create the following polynomial:

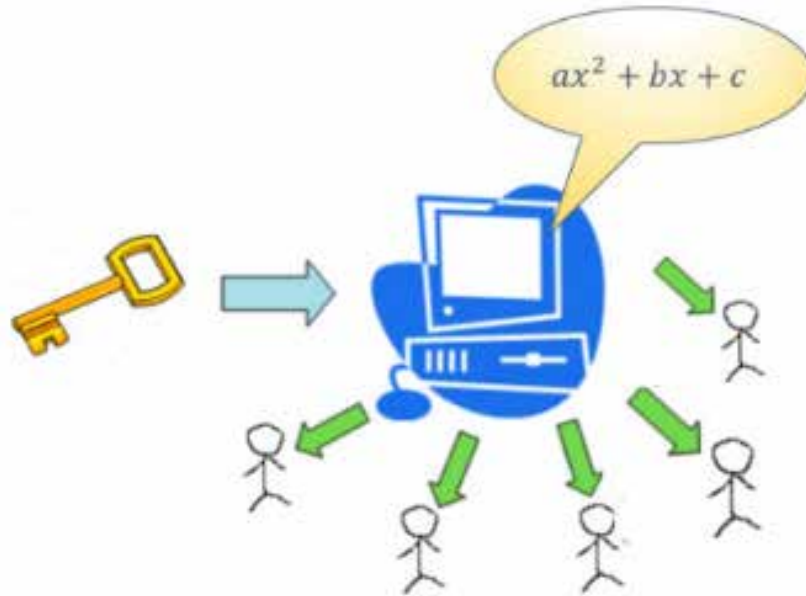
$$f(x) = a_0 + a_1x + a_2x^2 + \dots + a_{k-1}x^{k-1}$$

- Calculate the corresponding $y_i = f(x_i)$ values in the polynomial for different $x_i \pmod p$ values to distribute the key to the users. The resulting pairs can now be distributed to the users.

The set of users consists of $P = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ pairs. In the system built, any k user count finds the

$$f(x) = \sum_{i=1}^k y_i \prod_{i \neq j} \frac{x - x_j}{x_i - x_j} \pmod p$$

polynomial using the Lagrange interpolation formula and creating the key. Through the Lagrange interpolation formula, a polynomial with the degree of k-1 can be found by knowing the coordinates of k different points.



For example, let's create a system with Shamir's method, where the secret is distributed to 5 people, and it can't be found unless 3 people get together.

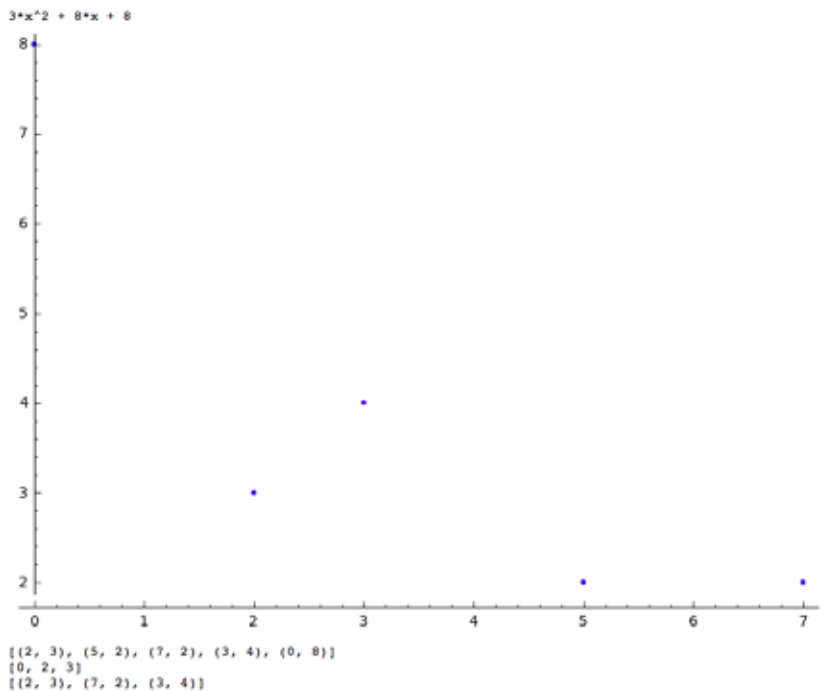
1. We accept as $S = 3 \pmod{11}$.

2. Let's create a polynomial $f(x) = x^2 + 2x + 3$ consisting of quadratic random numbers.

3. $P = \{(x_1, y_1), (x_2, y_2), \dots, (x_5, y_5)\}$ is the set of user, given to each user (x, y) pairs is calculated as:
 $5 \pmod{11}$; $x_4 = 6 \Rightarrow y_4 = 7 \pmod{11}$; $x_5 = 9 \Rightarrow y_5 = 3 \pmod{11}$

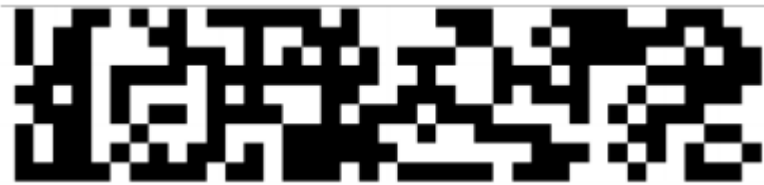
4. $P = \{P_1 = (1, 6), P_2 = (2, 0), P_3 = (5, 5), P_4 = (6, 7), P_5 = (9, 5)\}$ is a set of users consisting of pairs, where any 3 users can find the key by creating polynomial using Lagrange interpolation formula.

5. For example, $P = \{P_1, P_3, P_5\}$ users want to come together and find the secret. Using the Lagrange interpolation formula, the X polynomial is created and the key is found as $s = f(0) = 3$. The Lagrange interpolation formula is used here to form a second-degree polynomial known as the three different points it passes through.

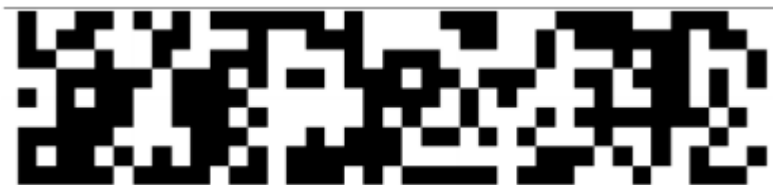


Visual Secret Sharing Systems

Let's briefly mention the Visual Secret Sharing System through an example. The interesting aspect of the visual secret sharing system is that the information shared is an image. In this system, which was introduced by Naor and Adi Shamir in 1994, the images are matrices consisting of black and white pixels. Black pixels are expressed with "1" and white pixels are expressed with "0". The following example shows the matrices created in this manner.



The first image is a picture obtained from random pixels.



The second image is an image obtained from the first image and a hidden image.

AЯKAKAPI

The hidden image will be revealed by adding the two matrices which correspond to the first and second image

Sources:

Shamir, A., (1979). "How to Share a Secret", Communications of the ACM, 22: 612-61

M. Naor and A. Shamir, 'Visual cryptography', Advanced in Cryptography- Eurocrypt94 ,vol.950, no.7, 1995, 1-12.

http://www.matematikdunyasi.org/arsiv/PDF/13_04_75_77_sir.pdf

<http://www.wikizeroo.net/index.php?q=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnL3dpa2kvU2hhbWlyJ3NfU2V-jcmV0X1N oYXJpbmc>

Database Attacks and Protection Methods

What first comes to mind when cybersecurity is mentioned are the web attacks, that is to say, external attacks. Let me confuse you a bit.

“What if the enemy is inside?”

That is to say, what if the person who is after your data is whom you drink coffee with every day?

However, life does not go on with this paranoia, a state of a vast incredulity nearly drifting one into becoming mad.

Of course, we won't suspect everyone, yet this does not prevent us from taking action.

Now, let's say that we have a database, let there be an MSSQL server. Here reside some very significant data of your company, such as finance, accounting, R&D, product design. These data may be subject to significant dangers; the data can be:

1. Deleted
2. Changed
3. Compromised by the rival company



Which one is more dangerous, or riskier changes depending on company and individuals. The risk is something not to take.

Database attacks can be made through various methods. However, I am going to mention the random password trying ways; the Brute Force attacks.

Usually, on a database system, it is expected that the system detects it when too many wrong passwords are entered.

The scene below is from the Şabanoglu Şaban movie of Kemal Sunal, where a password is asked.

The talk between the two is as follows:

-I am going to count to 3, tell the password right now! 1!

+Password. Umm!

The password is not "Umm". 2!

-Password. Stop, I'm going to find it!

-The password is neither "I'm going to find it"!

-Oh! I found it, "Başak".

-You didn't, and it's "Şafak"



This dialogue is an example of how it should be. If a person enters an incorrect password three times, he/she should be blocked.

Let's take a look at how things work in real life.

In an MSSQL database, you are allowed to enter an incorrect password as many times as you wish. That is to say; it does not detect if your actions are malicious. There exist no such setting relevant to this. If the system gives this opportunity, all that is left to the attacker is to try and find different combinations until the password is found.

Here are what you need to log into an MSSQL database.

1. Physical connection

2. For the 1433 port to be open

3. Username

4. Password

Let's think.

Our SQL server is not open to the internet, so the attacker can not attack from outside since there is no physical connection.

The SQL server can connect to the internet, but the port 1433 is closed; there is a physical connection, but the port is closed, so the access is once again denied.

The SQL is connected to the internet, and you opened the port 1433 to connect without VPN. In this case, the attacker should guess the username and password. Almost 99% of MSSQL users in Turkey do not disable the default admin account "SA". So what is left? To guess the password.

The password "safak" can be found using a 5-combination of the 26 characters of the English alphabet. That is to say, $26 \times 26 \times 26 \times 26 \times 26 = 11,881,376$ different password combinations.

Let's suppose that the attacker can try ten passwords per second over the internet, which means $11,881,376 / 10 = 1,188,137.6$ seconds, 19.802 minutes, 330 hours, 13.75 days.

Okay, so what will it be like when the attacker attacks from within?

In many businesses, there are computers with GBit connection speed and i5, i7 CPU.

In this case, you can try up to 5000 passwords a second, which means that - $11,881,376 / 5,000 = 2,376$ sec., 39 min.- you can crack the password in 39 minutes.

Right now we have the following question pop up in our minds: "Why would the password be five characters long?" right?

Right, it would not be. The time needed to crack a password would take a little longer than five characters, containing uppercase and lowercase letters. Maybe one day, maybe one week, maybe one month.

However, at the end of the day, in case of such attack, what us, the system administrators need to do is to:

1. Detect,

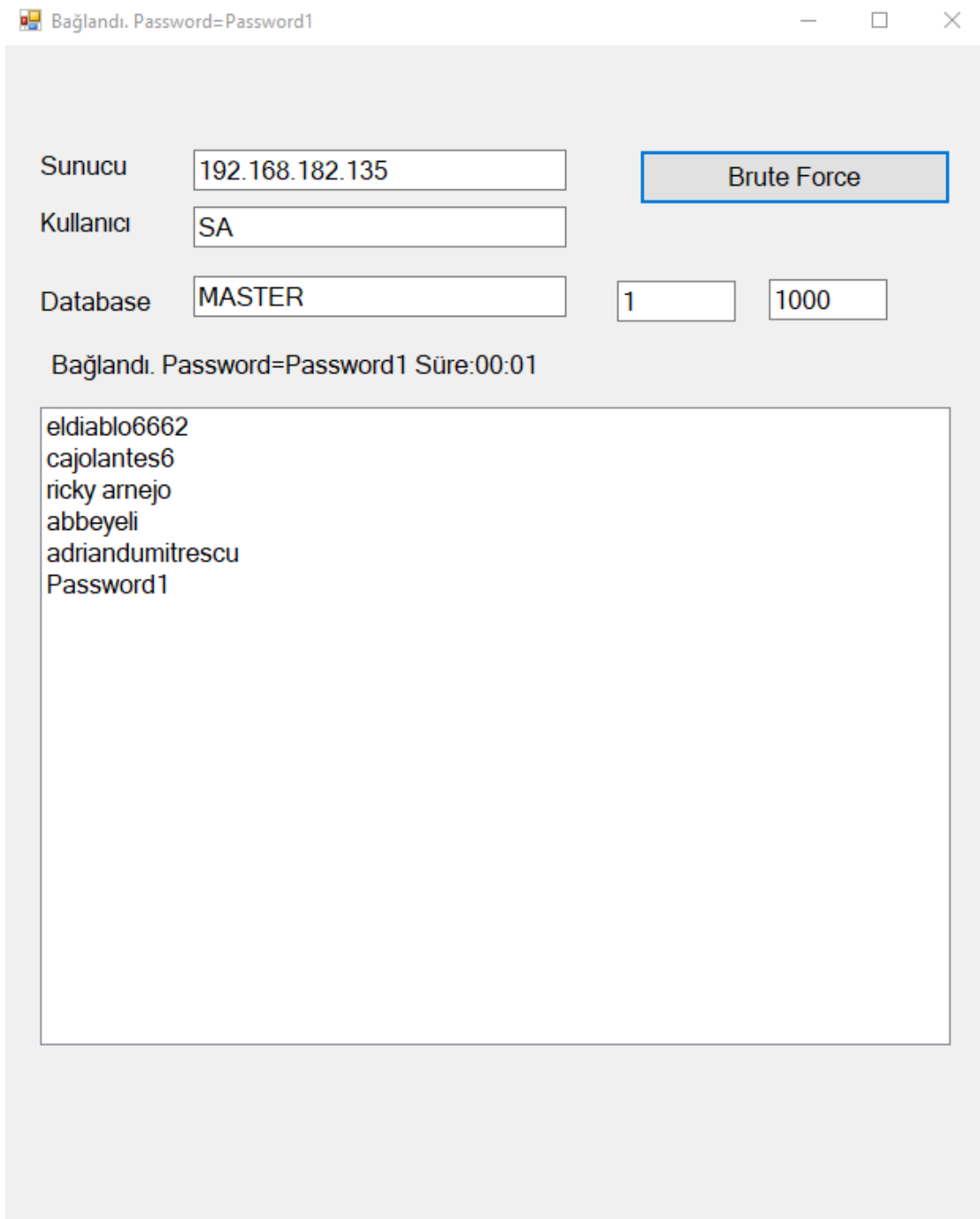
2. Record,

3. Block the attack
4. Distract and if possible catch the attacker on the job

In this article, I will be explaining how to do this.

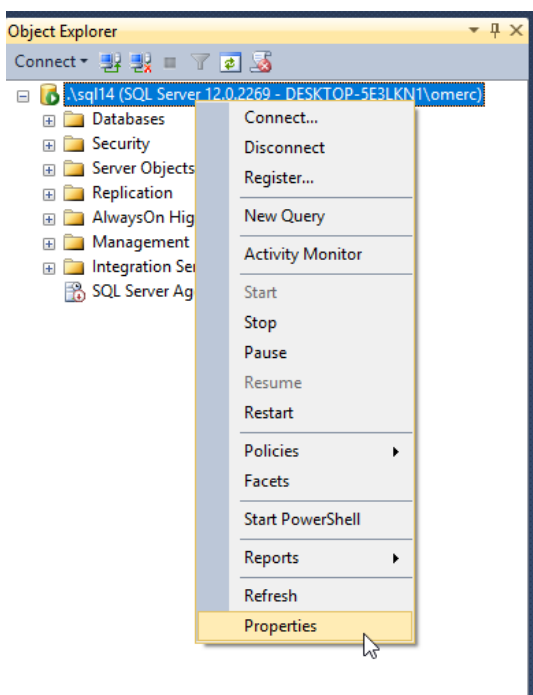
Below you see the picture of an application which performs continuous password attempts on the target system. The passwords it tries are not a combination, but rather a password dictionary of 2,000,000 passwords you can find online.

It is a simple application I wrote. It uses one thread and cracked the password by trying approximately 500 passwords a second.





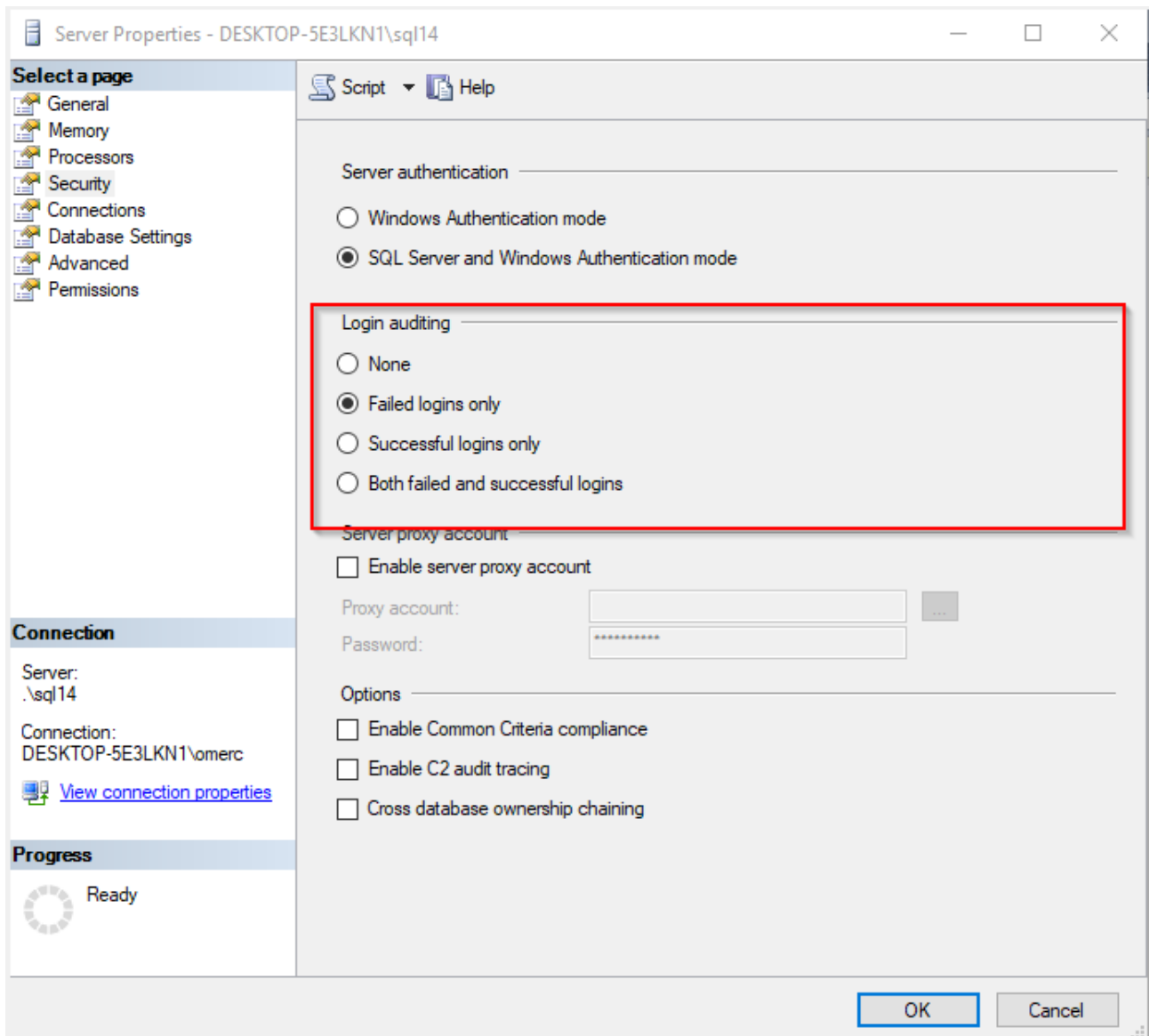
A screenshot from the password dictionary.



So, are we going to sit there while someone is trying to crack our password?

Of course not!

When you click on the Server Properties on the SQL Server, you can see a part named “Login Auditing”.



Here, we specify which login processes we would like to log.

Nothing is logged if None is chosen.

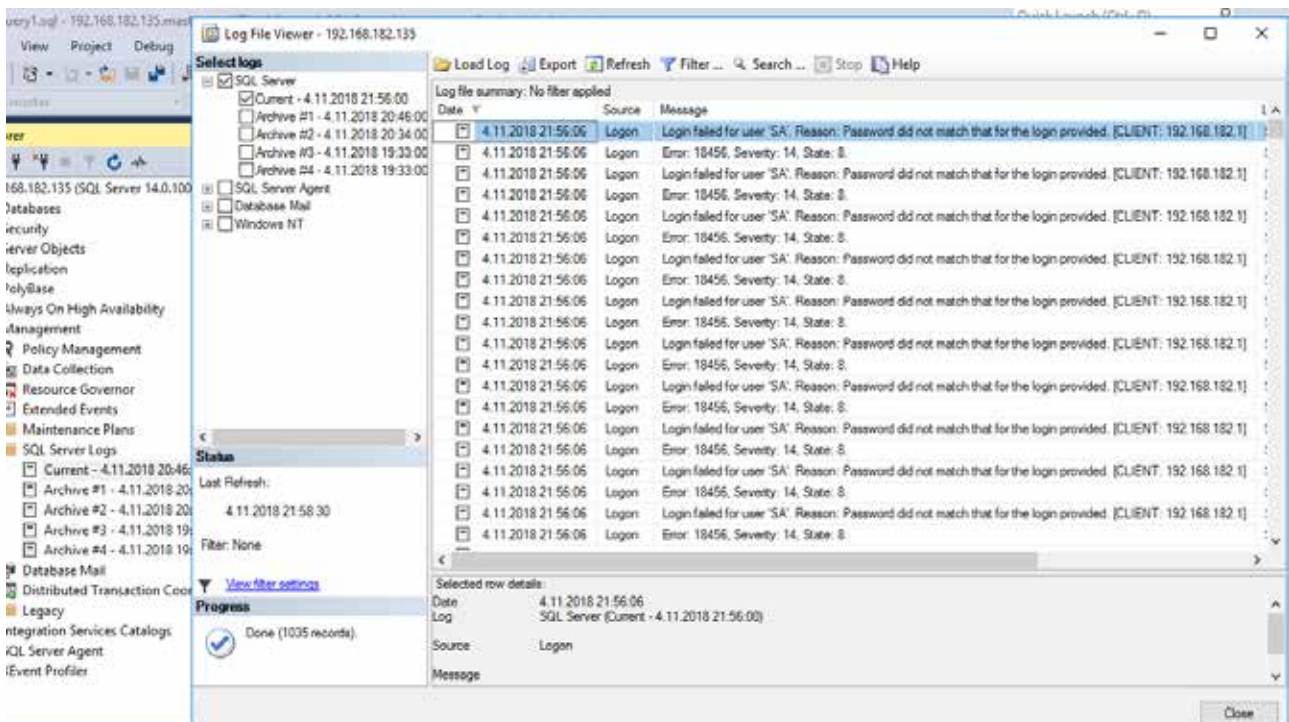
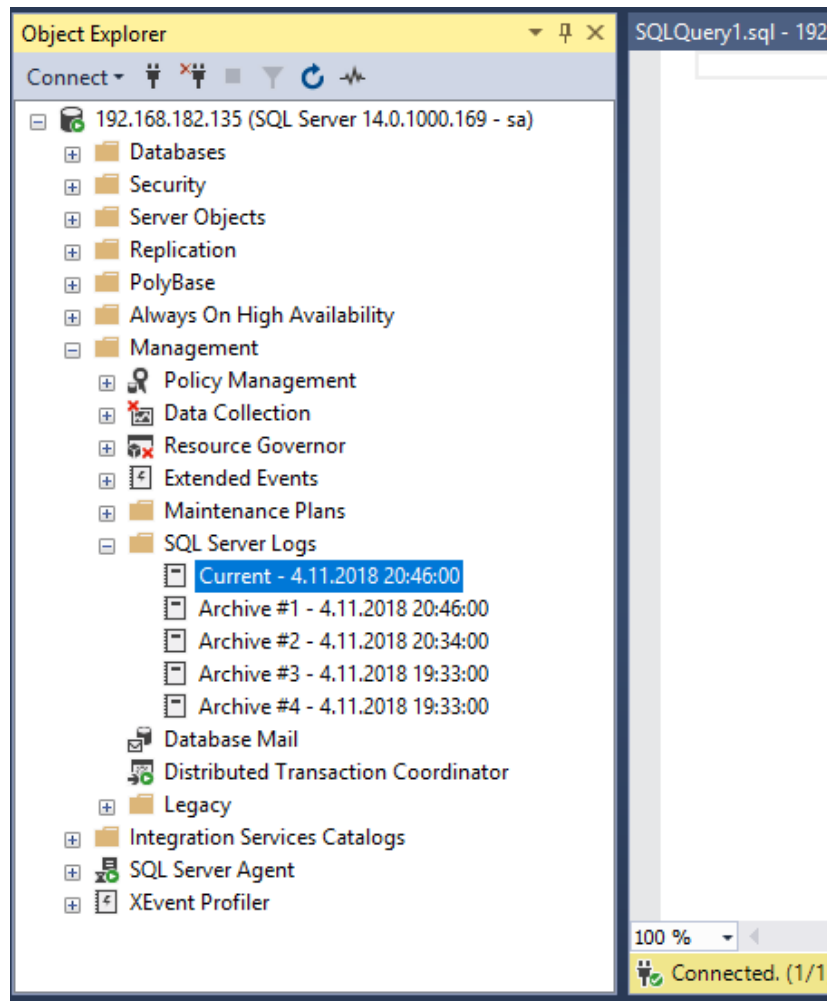
When Failed logins only option is chosen -which is the default option- just incorrect login attempts are logged.

In the Successful logins option, only successful entries are logged.

Both failed, and successful logins option logs them both.

Now let's see how we display these logs.

To see these logs, one needs to go to the SQL Management Studio Management part, enter the SQL Server Logs tab and click on the Current Logs menu.



It can be seen that there a lot of “Login Failed for user ‘SA’ Reason: Password did not match that for the login provided. [CLIENT:192.168.182.1]” log.

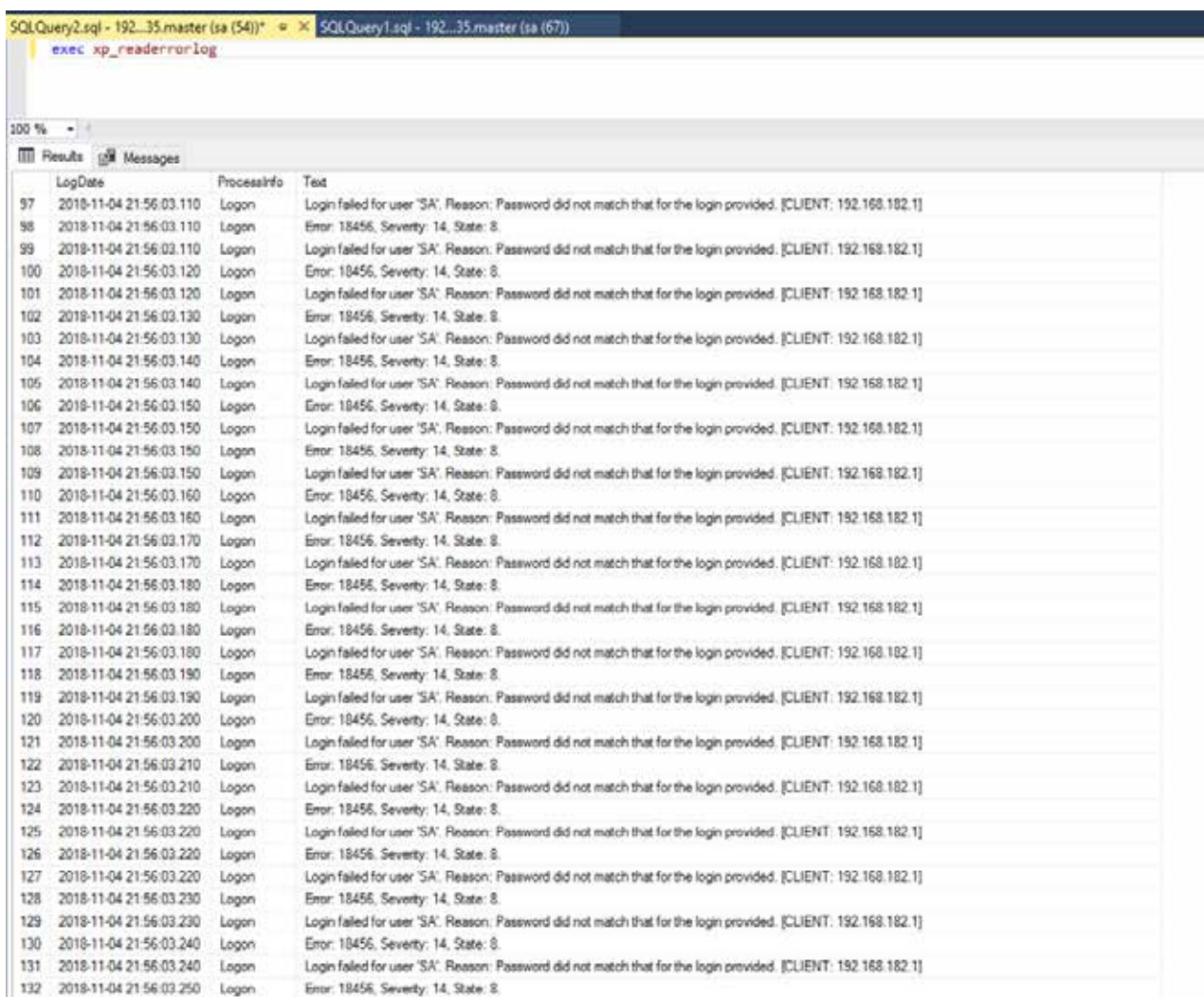
This case shows us that a password trying attacks has been made. Furthermore, we also have the IP address the attack is made from.

So, to notice this attack, do we need to look at this screen continually?

In the background, the SQL Server understands nothing but the SQL language.

That is to say, to create the log table we see, there is an SQL query made in the background.

“EXEC XP_ReadErrorLog”



LogDate	ProcessInfo	Text
2018-11-04 21:56:03.110	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-04 21:56:03.110	Logon	Error: 18456, Severity: 14, State: 8.
2018-11-04 21:56:03.110	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-04 21:56:03.120	Logon	Error: 18456, Severity: 14, State: 8.
2018-11-04 21:56:03.120	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-04 21:56:03.130	Logon	Error: 18456, Severity: 14, State: 8.
2018-11-04 21:56:03.130	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-04 21:56:03.140	Logon	Error: 18456, Severity: 14, State: 8.
2018-11-04 21:56:03.140	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-04 21:56:03.150	Logon	Error: 18456, Severity: 14, State: 8.
2018-11-04 21:56:03.150	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-04 21:56:03.150	Logon	Error: 18456, Severity: 14, State: 8.
2018-11-04 21:56:03.150	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-04 21:56:03.160	Logon	Error: 18456, Severity: 14, State: 8.
2018-11-04 21:56:03.160	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-04 21:56:03.170	Logon	Error: 18456, Severity: 14, State: 8.
2018-11-04 21:56:03.170	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-04 21:56:03.180	Logon	Error: 18456, Severity: 14, State: 8.
2018-11-04 21:56:03.180	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-04 21:56:03.180	Logon	Error: 18456, Severity: 14, State: 8.
2018-11-04 21:56:03.180	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-04 21:56:03.190	Logon	Error: 18456, Severity: 14, State: 8.
2018-11-04 21:56:03.190	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-04 21:56:03.200	Logon	Error: 18456, Severity: 14, State: 8.
2018-11-04 21:56:03.200	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-04 21:56:03.210	Logon	Error: 18456, Severity: 14, State: 8.
2018-11-04 21:56:03.210	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-04 21:56:03.220	Logon	Error: 18456, Severity: 14, State: 8.
2018-11-04 21:56:03.220	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-04 21:56:03.220	Logon	Error: 18456, Severity: 14, State: 8.
2018-11-04 21:56:03.220	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-04 21:56:03.230	Logon	Error: 18456, Severity: 14, State: 8.
2018-11-04 21:56:03.230	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-04 21:56:03.240	Logon	Error: 18456, Severity: 14, State: 8.
2018-11-04 21:56:03.240	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-04 21:56:03.250	Logon	Error: 18456, Severity: 14, State: 8.

Now that we have a command let's suppose a scenario. Let the system:

- Work every 3 minutes
- Read the relevant log lines
- Detect that there is an attack if there are more then 100 logs containing the sentence “Login Failed for user ‘SA’ Reason: Password did not match that for the login provided. “

Here, 100 is a symbolic number. You may as well make it less or more.

All right, so how do we filter a prepared query like “EXEC XP_ReadErrorLog”?

Quickly, we can do it using the Temp Table. Temp Tables are tables which are created in the memory and disappear later.

CREATE TABLE #SQLErrorLog

```
(
    LogDate DATETIME ,
    ProcessInfo VARCHAR(20) ,
    Text VARCHAR(500)
);
```

INSERT INTO #SQLErrorLog

```
EXEC xp_readerrorlog 0;
```

SELECT * FROM #SQLErrorLog

```
WHERE Text LIKE '%Login failed for user%'
```

```
AND LogDate >= DATEADD(MINUTE, -1 * 3, GETDATE())
```

As can be seen, 476 password attempts had been made the last 3 minutes.

The screenshot shows a SQL Server query window with the following code:

```
CREATE TABLE #SQLErrorLog
(
    LogDate DATETIME ,
    ProcessInfo VARCHAR(20) ,
    Text VARCHAR(500)
);

INSERT INTO #SQLErrorLog
EXEC xp_readerrorlog 0;

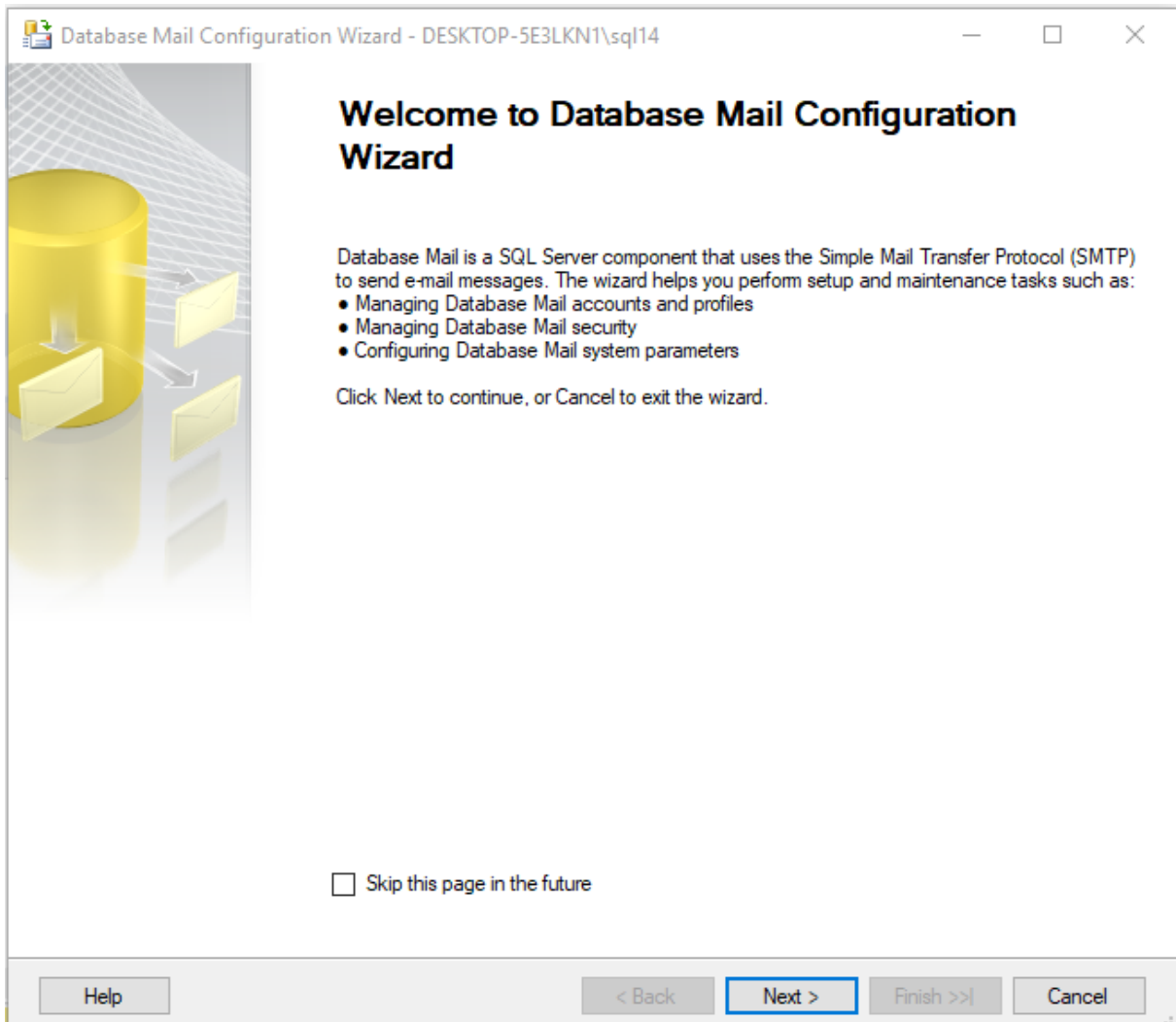
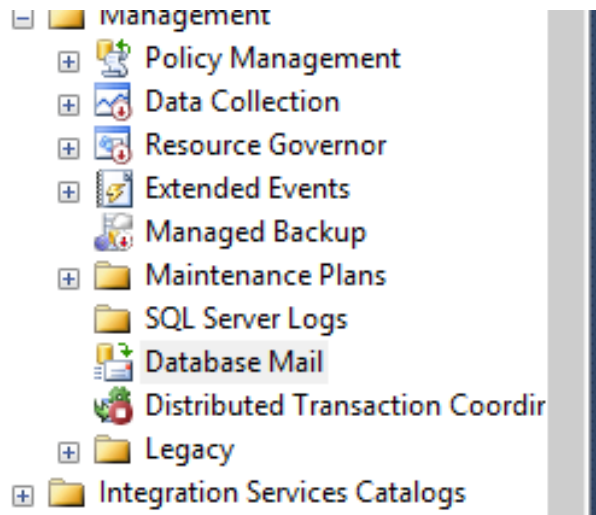
SELECT * FROM #SQLErrorLog
WHERE Text LIKE '%Login failed for user%'
AND LogDate >= DATEADD(MINUTE, -1 * 3, GETDATE())
```

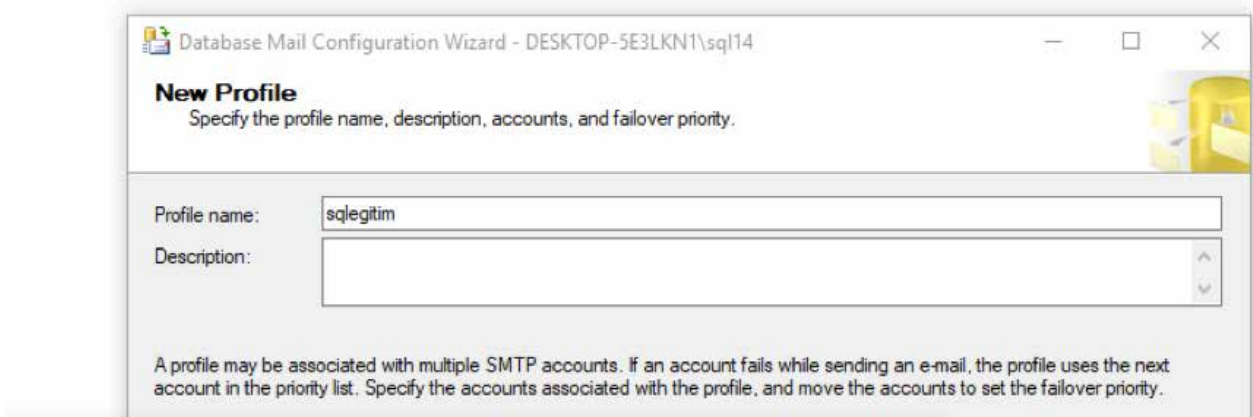
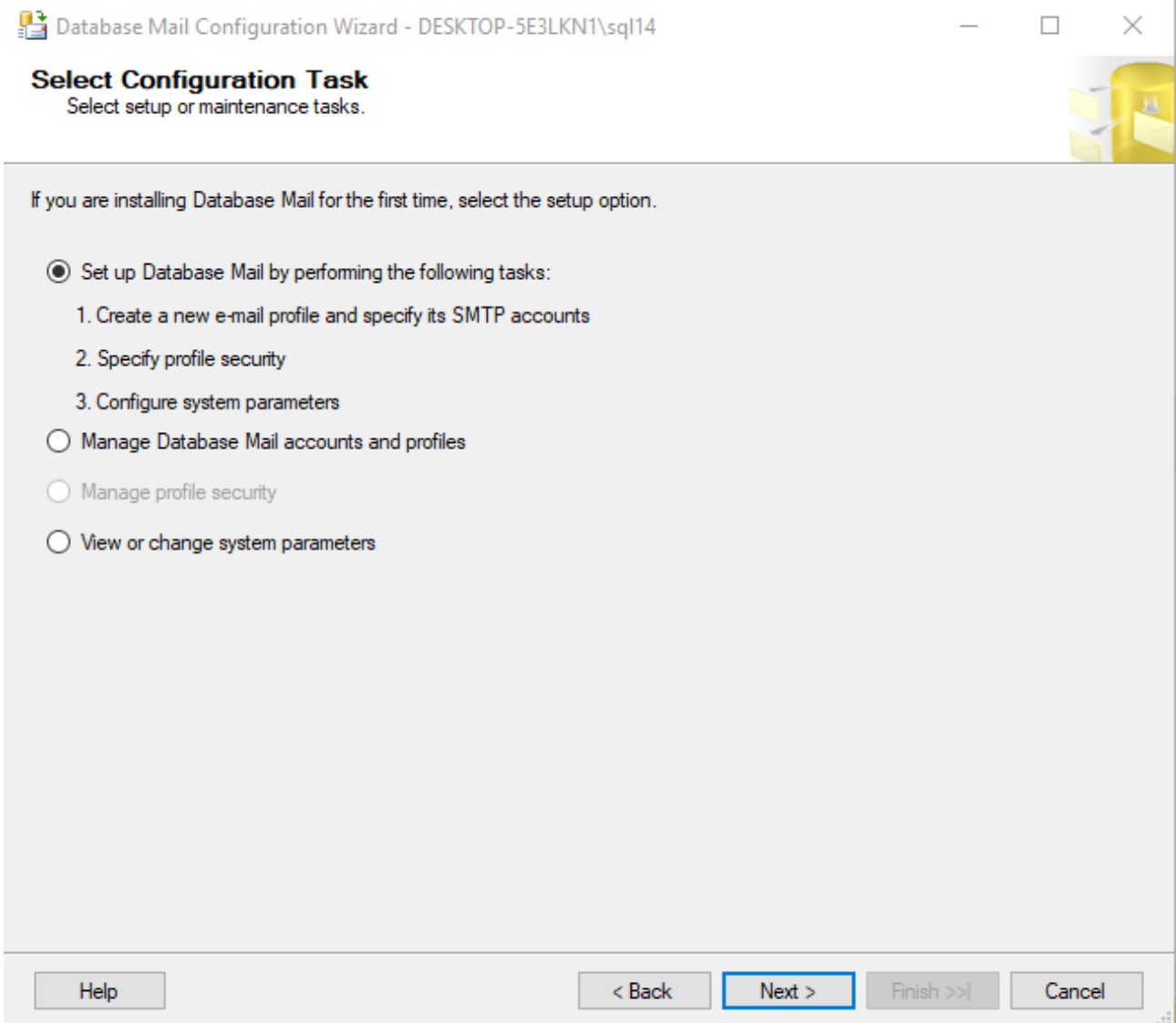
The results window shows a table with the following columns: LogDate, ProcessInfo, and Text. The data consists of 476 rows, all showing login failures for user 'SA'.

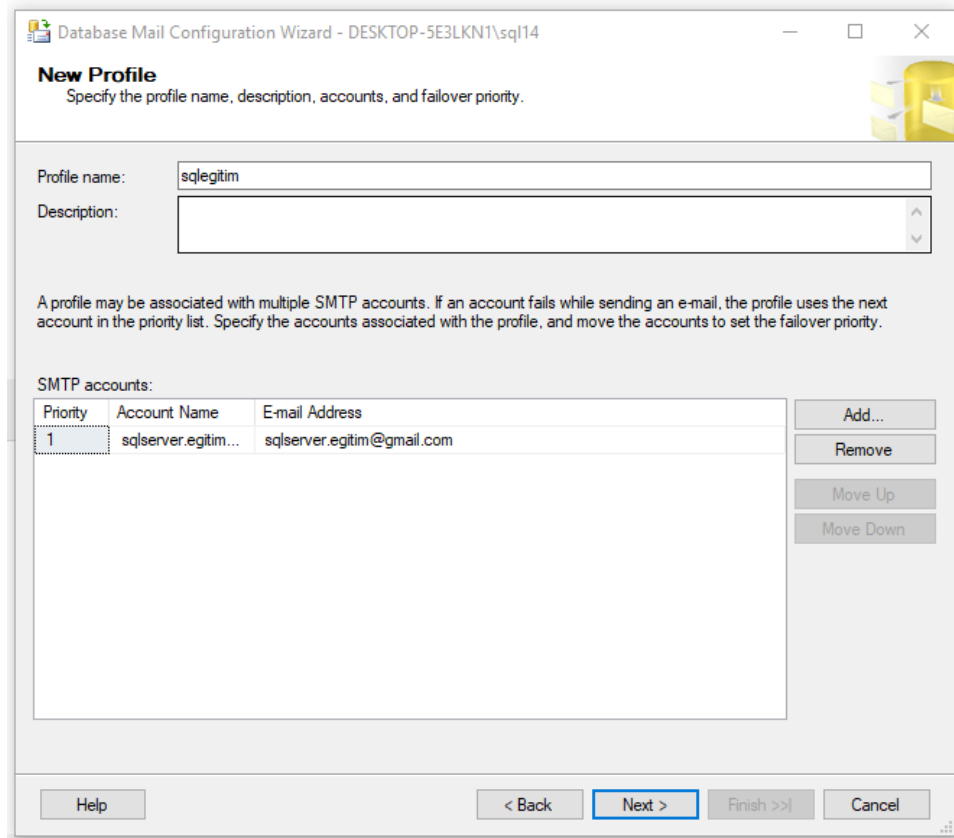
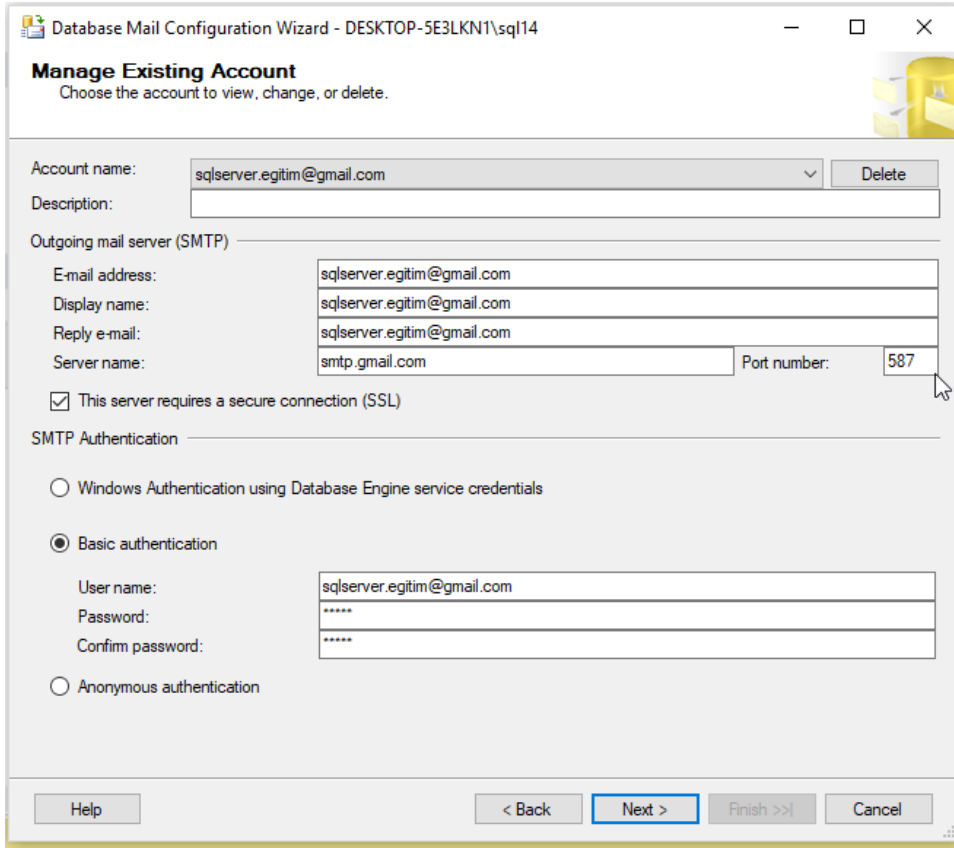
LogDate	ProcessInfo	Text
2018-11-07 01:11:13.820	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-07 01:11:13.830	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-07 01:11:13.840	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-07 01:11:13.850	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-07 01:11:13.860	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-07 01:11:13.870	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-07 01:11:13.880	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-07 01:11:13.890	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-07 01:11:13.900	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]
2018-11-07 01:11:13.910	Logon	Login failed for user 'SA'. Reason: Password did not match that for the login provided. [CLIENT: 192.168.182.1]

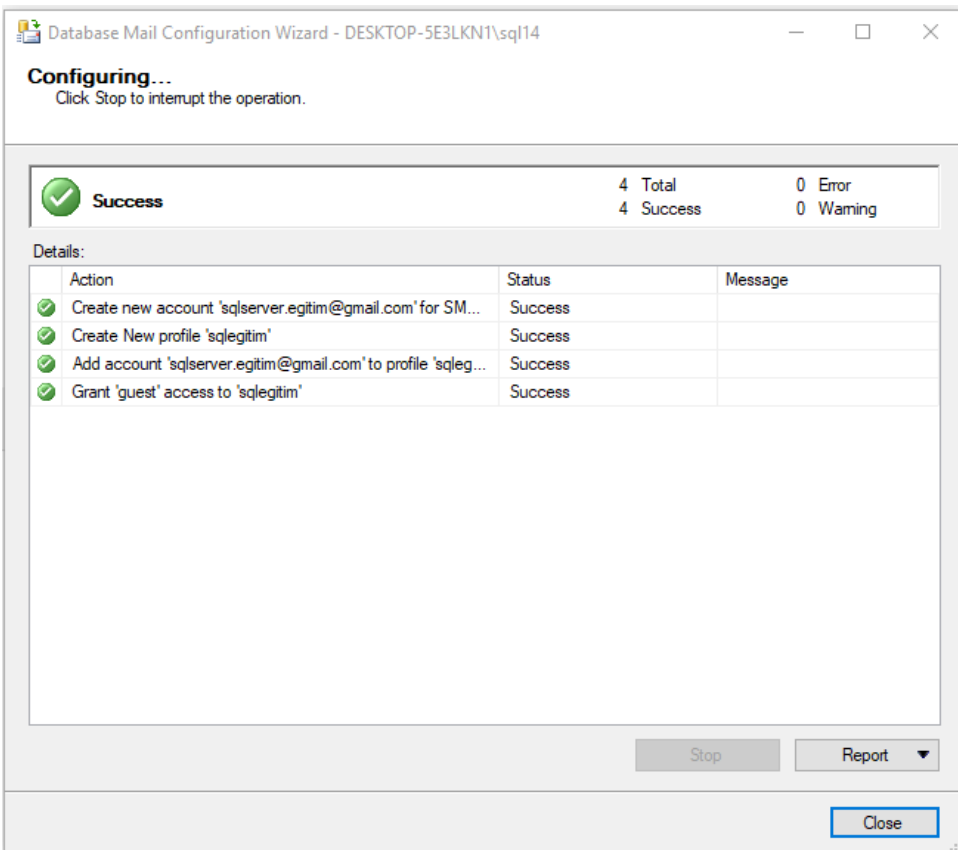
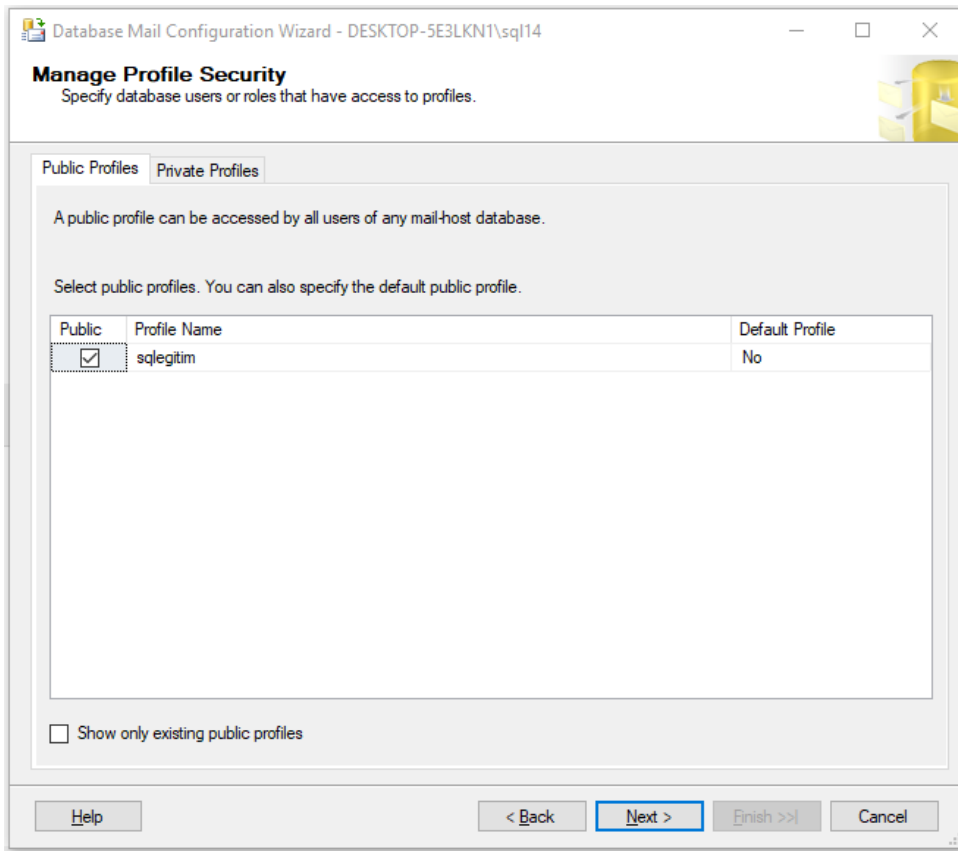
The status bar at the bottom indicates: Query executed successfully. 192.168.182.138 (14.0 RTM) SA (56) master 00:00:00 476 rows

Now, what we need to do here is to detect the IP address and inform the database manager. To do this, DB Mail in the SQL Server needs to be activated. SQL DB Mail installation can be done as follows:

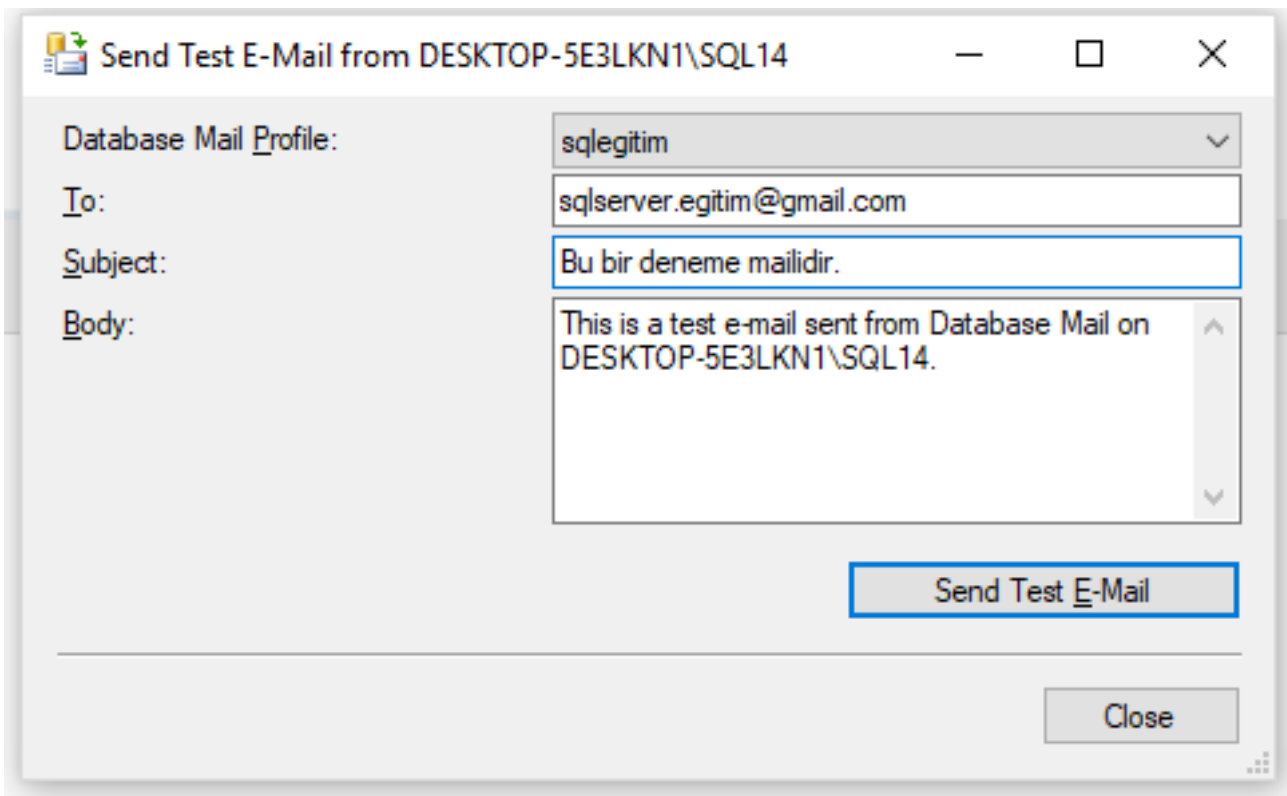
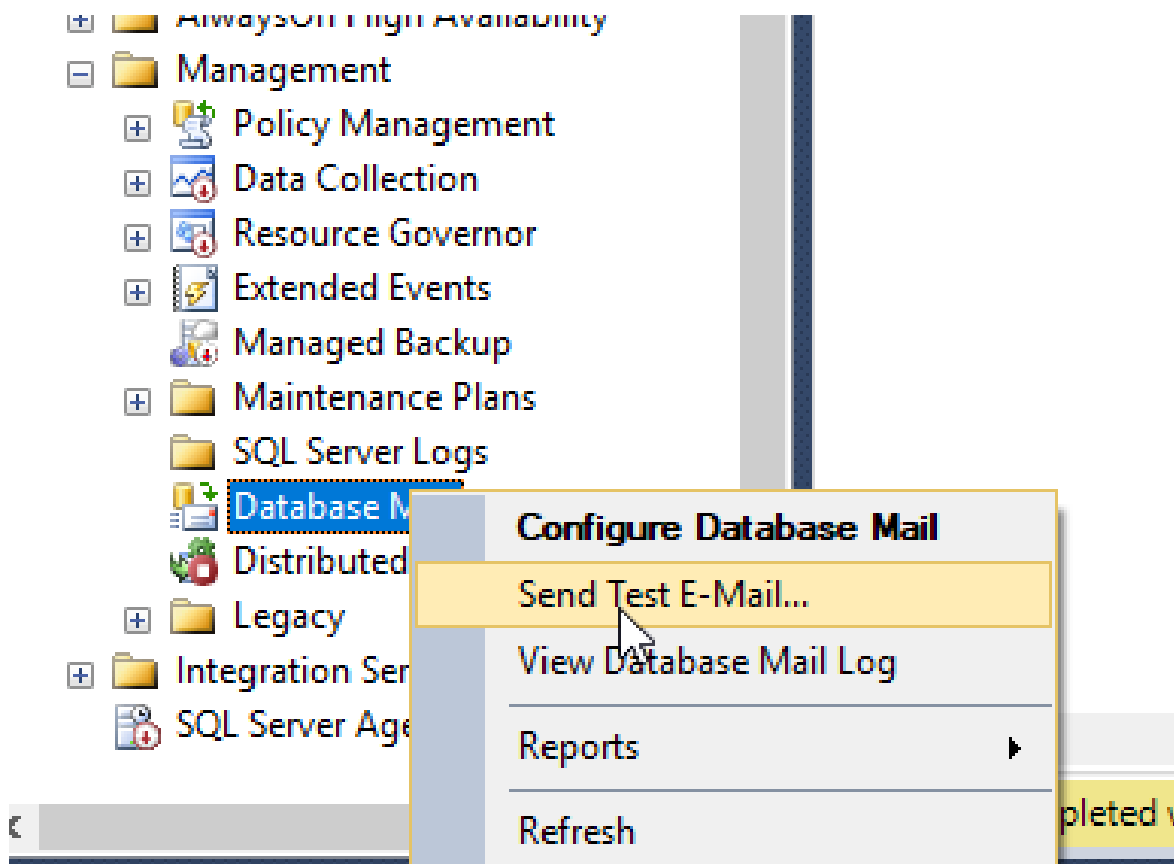




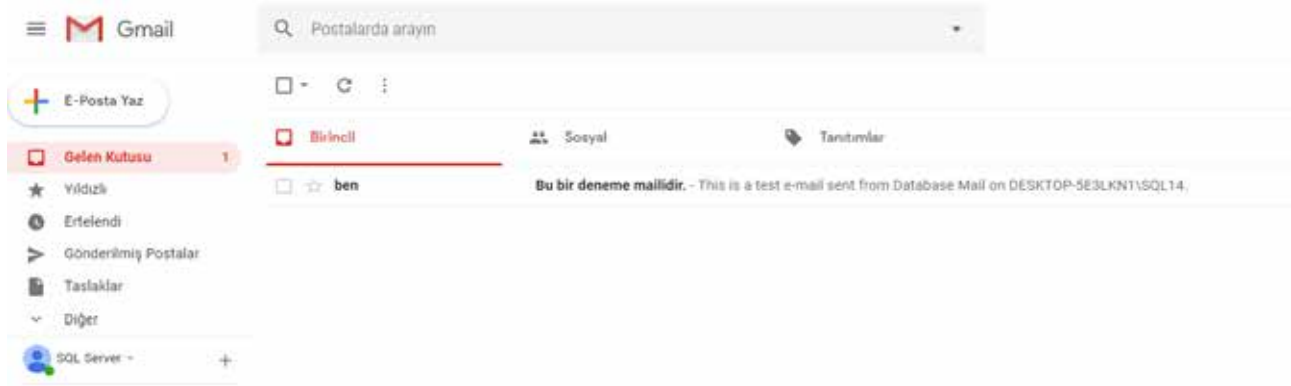




Now, let's send a test email.



The mail we sent is received by the SQL Server.



EXEC msdb.dbo.sp_send_dbmail

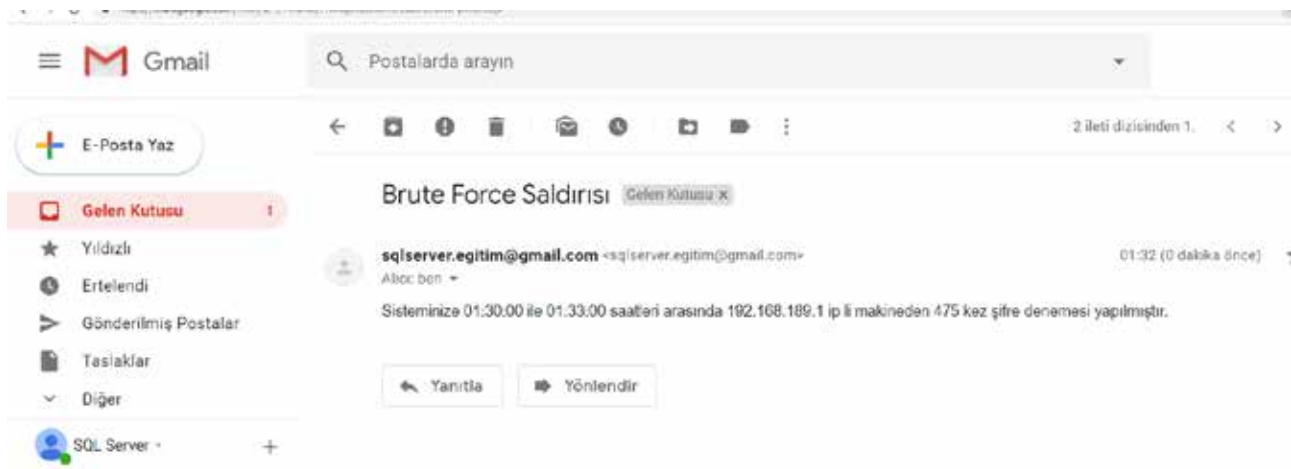
@profile_name = 'sqlegitim',

@recipients = 'sqlserver.egitim@gmail.com',

@subject = 'Brute Force Saldırısı',

@body='Sisteminize 01:30:00 ile 01:33:00 saatleri arasında 192.168.189.1 ip li makineden 475 kez Şifre denemesi yapılmıştır.'

After activating the mail system, we need to send a mail to the database manager with a meaningful message. The mail sending process is again actually a T-SQL command.



Let's write a query which will do this operation automatically.

```
CREATE PROC BRUTFORCE_CONTROL @MINUTE AS INT = 3
AS
    BEGIN
CREATE TABLE #SQLErrorLog
    (
        LogDate DATETIME ,
        ProcessInfo VARCHAR(20) ,
        Text VARCHAR(500)
    );

INSERT INTO #SQLErrorLog
    EXEC xp_readerrorlog 0;

    DECLARE @TEXT AS VARCHAR(1000);
    DECLARE @COUNT AS INT;
    DECLARE @MINDATE AS DATETIME;
    DECLARE @MAXDATE AS DATETIME;

    SELECT @TEXT = Text ,
           @COUNT = COUNT(*) ,
           @MINDATE = MIN(LogDate) ,
           @MAXDATE = MAX(LogDate)
    FROM #SQLErrorLog
    WHERE ( Text LIKE '%Login failed for user%' )
           AND LogDate >= DATEADD(MINUTE, -1 * @MINUTE, GETDATE())
    GROUP BY Text
    HAVING COUNT(*) > 5;

    DECLARE @USER AS VARCHAR(100);
    DECLARE @IP AS VARCHAR(100);

--ORDER BY LogDate DESC
```

```

DECLARE @POS AS INT= 0;
DECLARE @POS2 AS INT= 0;

```

```

DECLARE @STR1 AS VARCHAR(1000)= REPLACE(@TEXT, 'Login failed for USER',
                                          '');

```

```

SET @POS = CHARINDEX('. Reason:', @STR1);
SET @USER = LEFT(@STR1, @POS - 1);

```

```

SET @POS = CHARINDEX('[CLIENT: ', @TEXT);
SET @POS2 = CHARINDEX(']', @TEXT);

```

```

SET @IP = SUBSTRING(@TEXT, @POS, @POS2 - @POS);
SET @IP = REPLACE(@IP, '[CLIENT: ', '');

```

```

DECLARE @MSG AS VARCHAR(1000);
SET @MSG = CONVERT(VARCHAR, @MINDATE, 109) + ' VE '
+ CONVERT(VARCHAR, @MAXDATE, 109) + '

```

```

TARİHLERİ ARASINDA ' + @USER
+ ' KULLANICISI ' + CONVERT(VARCHAR, @COUNT)
+ ' KEZ YANLIŞ ŞİFRE GİRDİ';

```

```

IF @COUNT > 5
  BEGIN
    EXEC msdb.dbo.sp_send_dbmail
      @profile_name = 'sqlegitim',
      @recipients = 'sqlserver.egitim@gmail.com',
      @subject = 'Brute Force Saldırısı',
      @body=@MSG
  END;
SELECT @MSG;
DROP TABLE #SQLErrorLog;

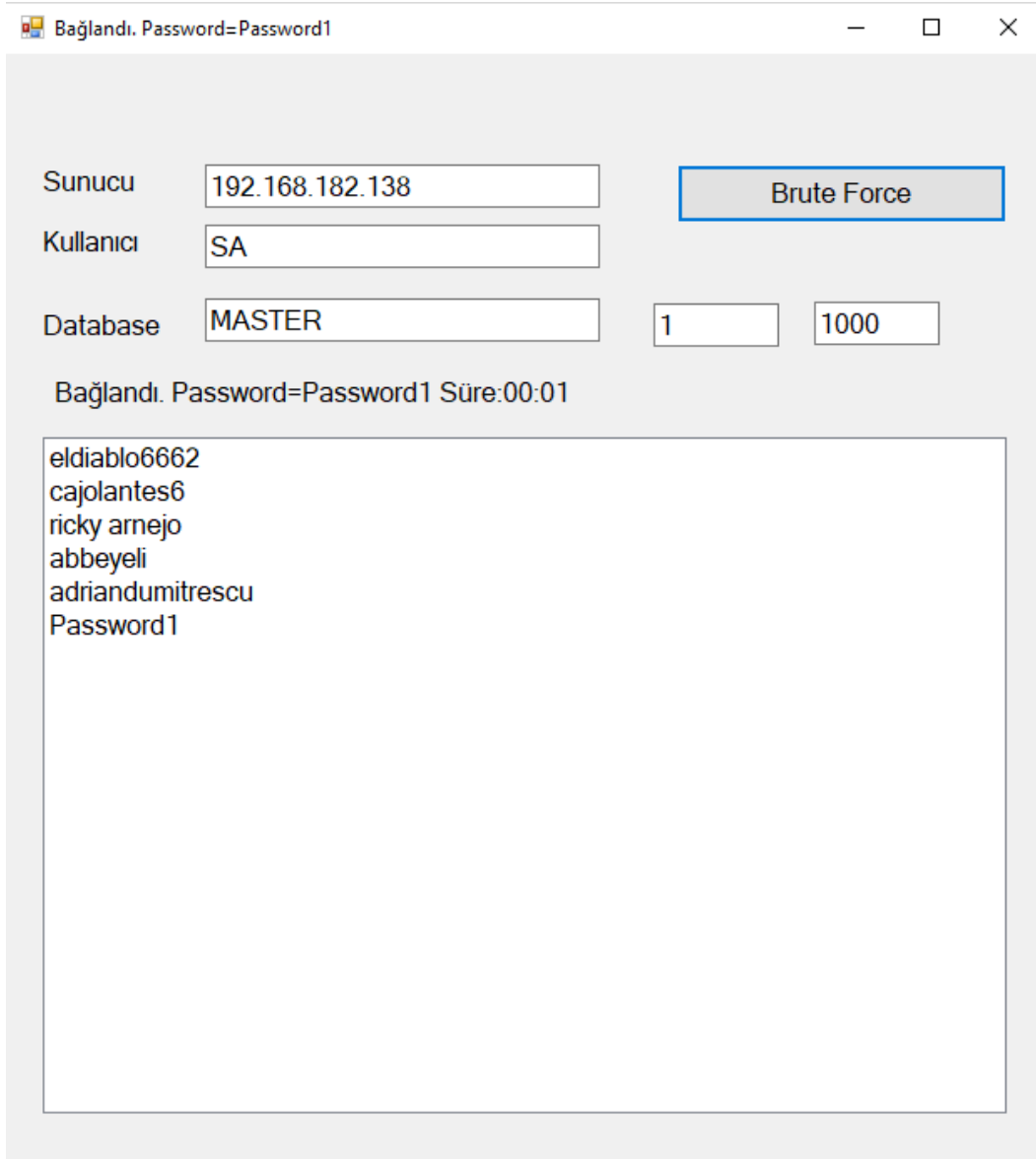
```

```

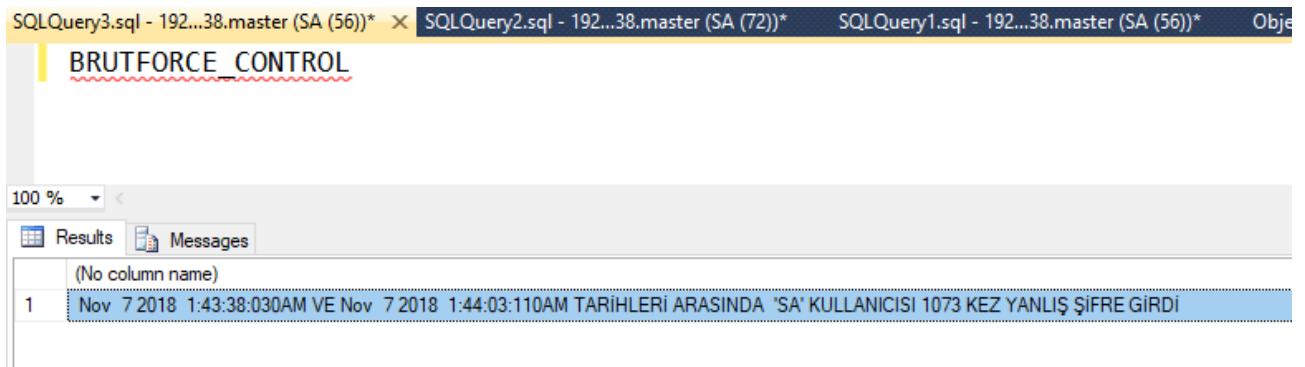
END;

```


Now let's do a brute force attack on the system several times.

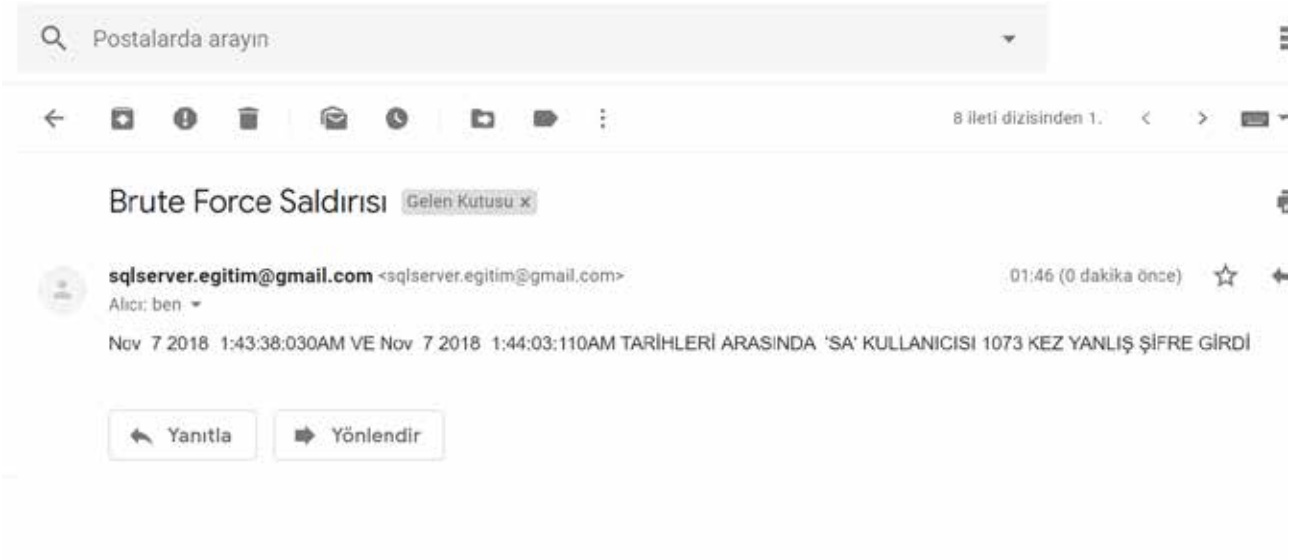


Run the BRUTFORCE_CONTROL query manually.



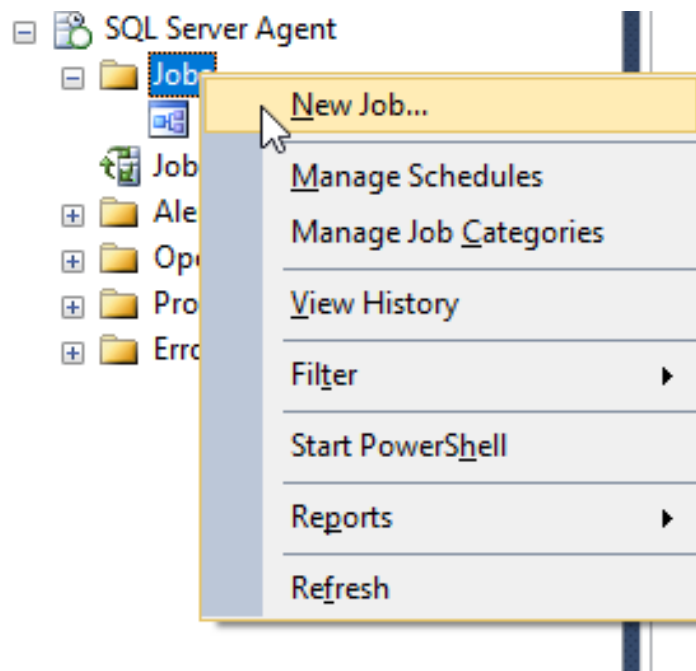
The response of the query came like this.

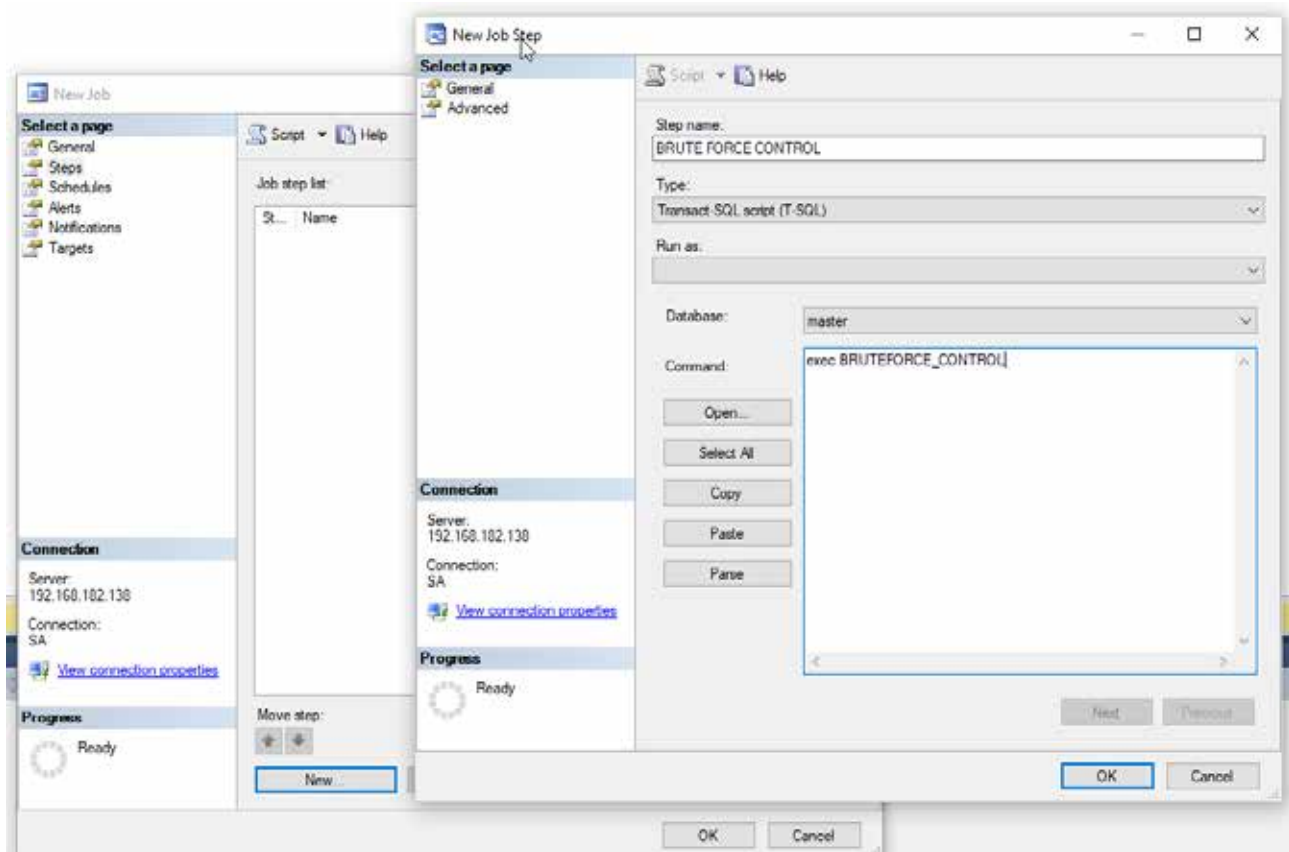
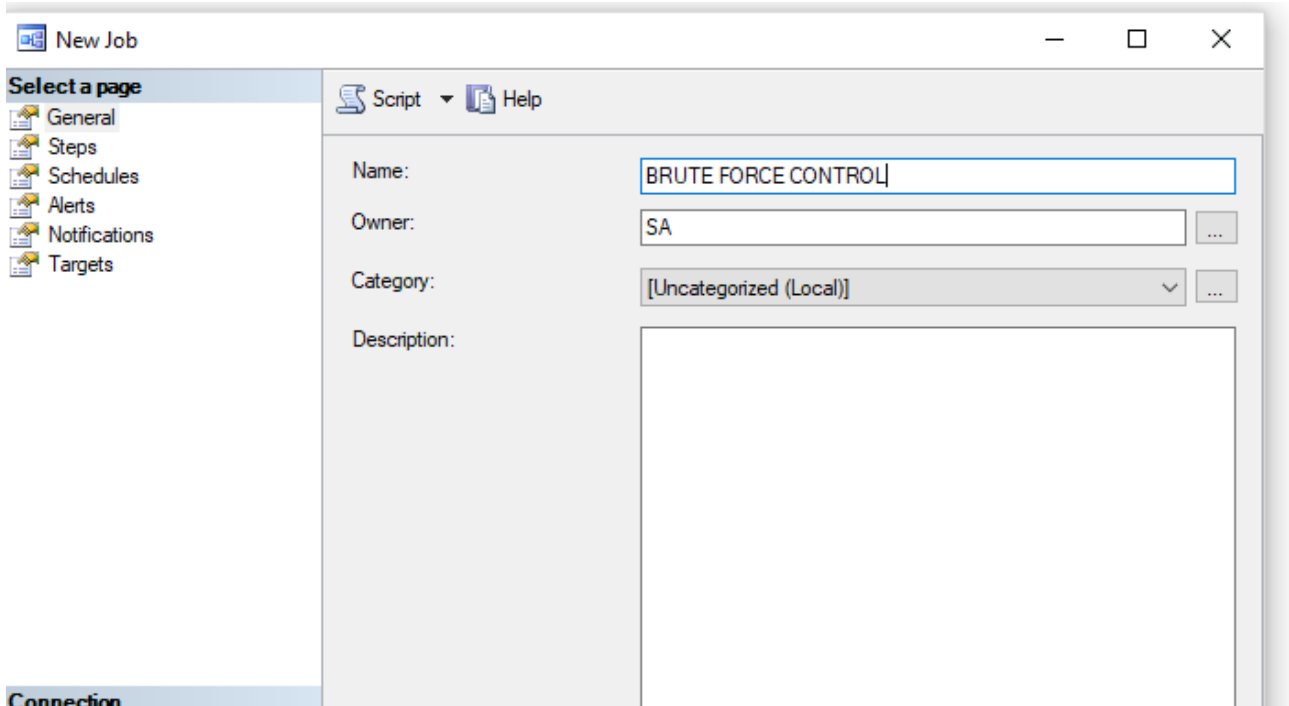
Below, the mail received can be seen.

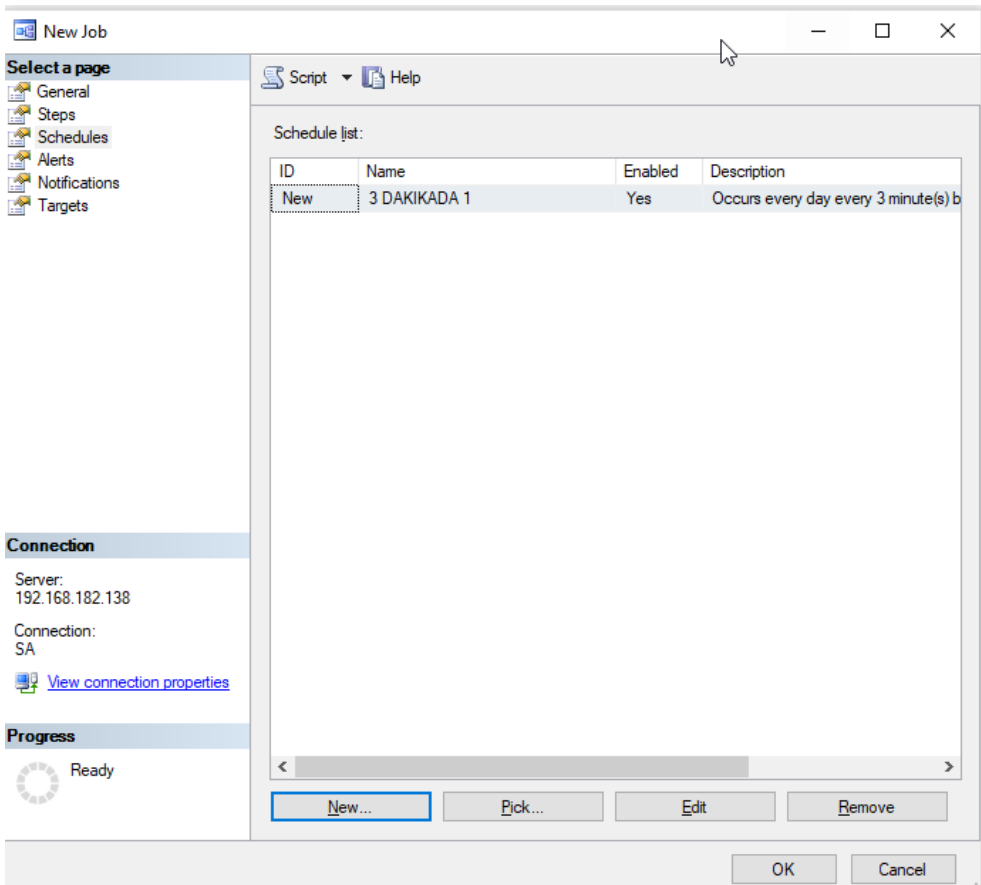
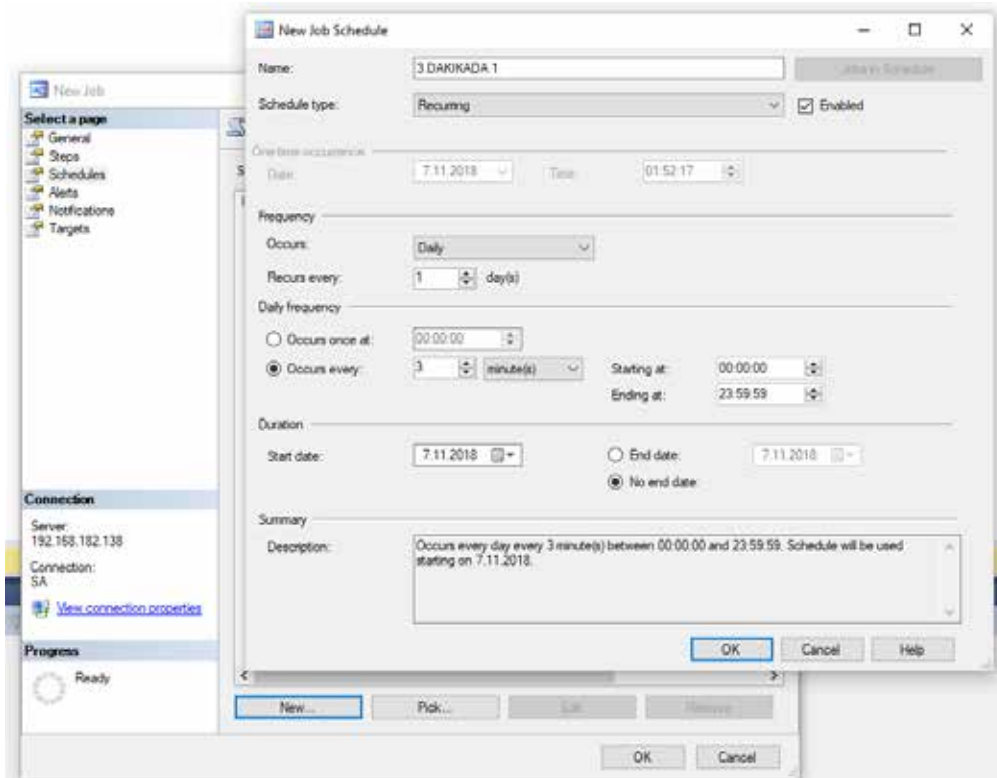


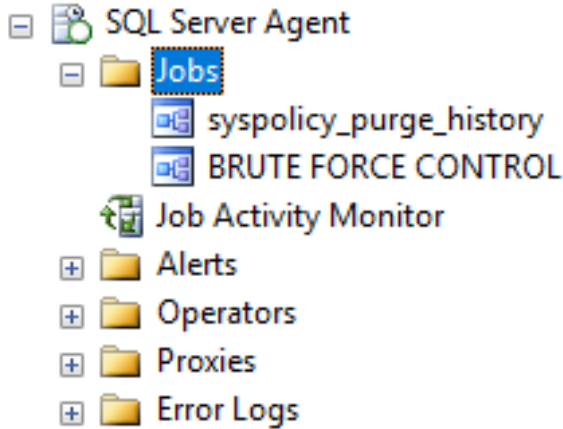
What comes next is to do this operation automatically rather than manually.

In order to do this, writing a job to the SQL Server Agent is the most practical way.

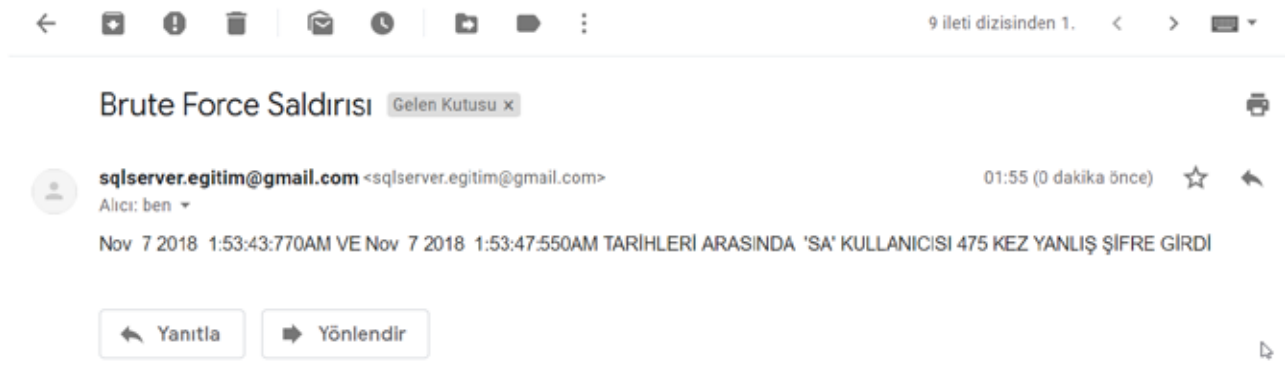








When we attack again, it can be seen that the system automatically sends us a mail in 3 minutes.



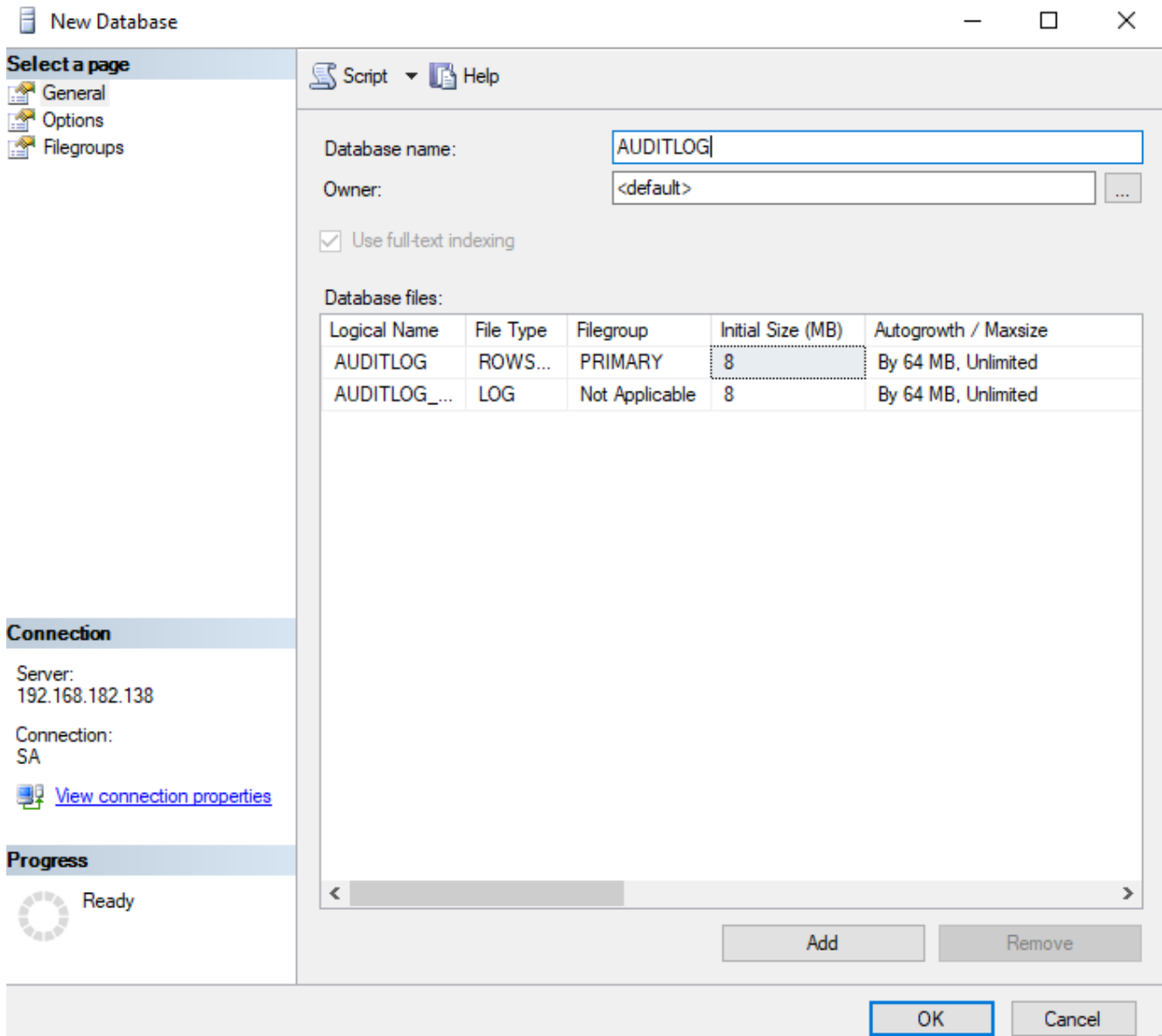
Now that we have detected the attack automatically, all we need to do is to block the entries coming from this IP address automatically.

You may as well write a rule with Powershell for Windows Firewall, or more efficiently, you can use the Server Logon Triggers.

It is indeed logical: now that we got the IP address of the attacker, we might as well create a blacklist table and write this IP within. Afterward, let us write a Logon Trigger and if the IP exists in this blacklist table, do not even allow him/her enter the password even if one knows it.

For this, we are going to make a change in a Stored Procedure named BRUTEFORCE_CONTROL.

First of all, let's create a database named AUDITLOG.



Create a table named blacklist in this database.

```
CREATE TABLE [dbo].[BLACKLIST](
    [ID] [int] IDENTITY(1,1) NOT NULL,
    [DATE_] [datetime] NULL,
    [IPADDRESS] [varchar](50) NULL
)
```

Now, update the Stored Procedure named BRUTEFORCE_CONTROL as follows, and add this code right before sending an email.

```
INSERT INTO AUDITLOG.dbo.BLACKLIST
```

```
    ( IPADDRESS,  
      DATE_)
```

```
VALUES ( @IP ,  
        GETDATE())
```

Finally, write a server trigger so that an IP found in this blacklist cannot enter the system.

However, before doing this operation, I highly recommend you to backup the system database, since in case of an error, neither you can enter the SQL Server.

Backup the master.mdf and master.ldf files inside the C:\Program Files\Microsoft SQL Server\MSSQL12\MSSQL\DATA directory. You need to stop the SQL service to be able to backup.

Later, you block the entry of the computers in the blacklist by executing the query below.

```
create TRIGGER [connection_limit_trigger]
```

```
ON ALL SERVER
```

```
FOR LOGON
```

```
AS
```

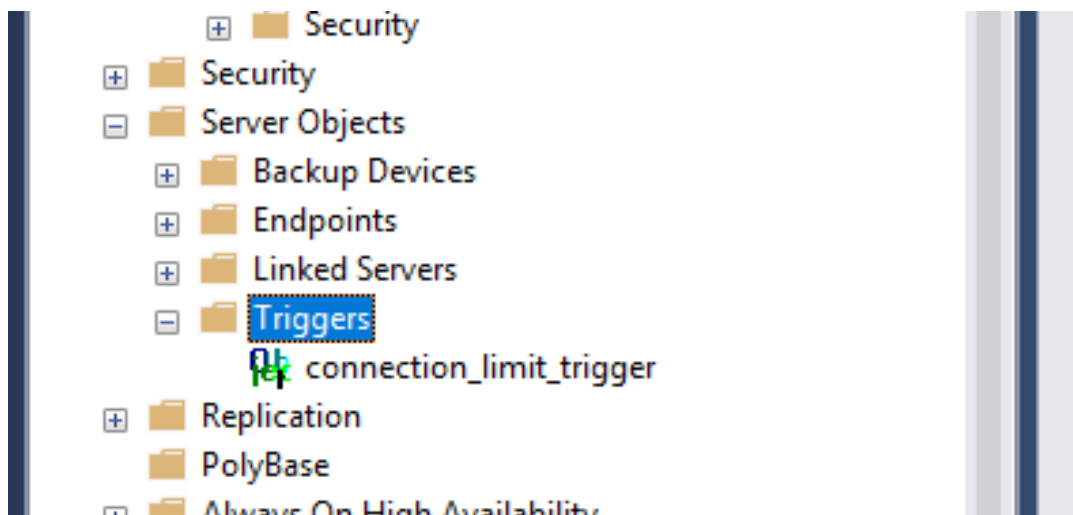
```
BEGIN
```

```
IF CONNECTIONPROPERTY ('client_net_address') IN (SELECT IPADDRESS FROM AUDIT-  
LOG.DBO.BLACKLIST)
```

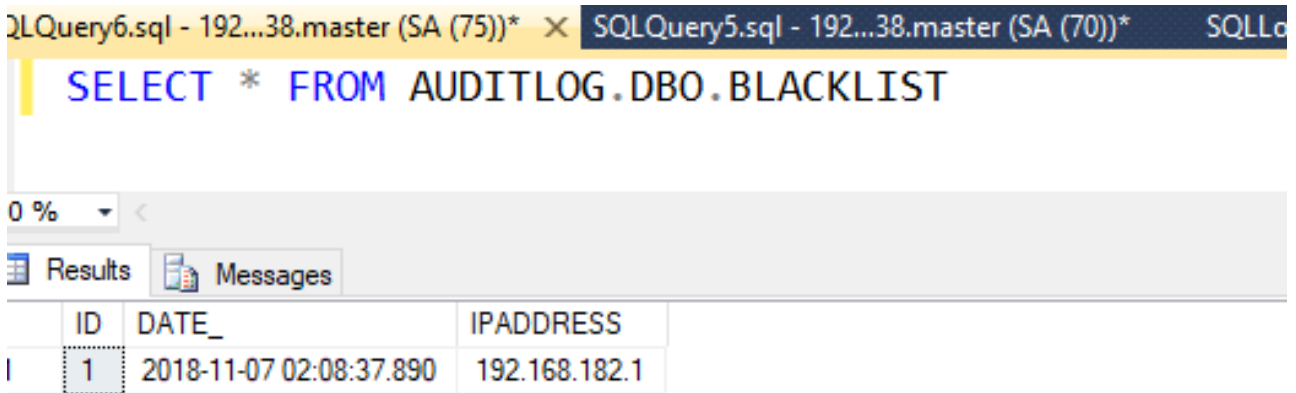
```
ROLLBACK;
```

```
END
```

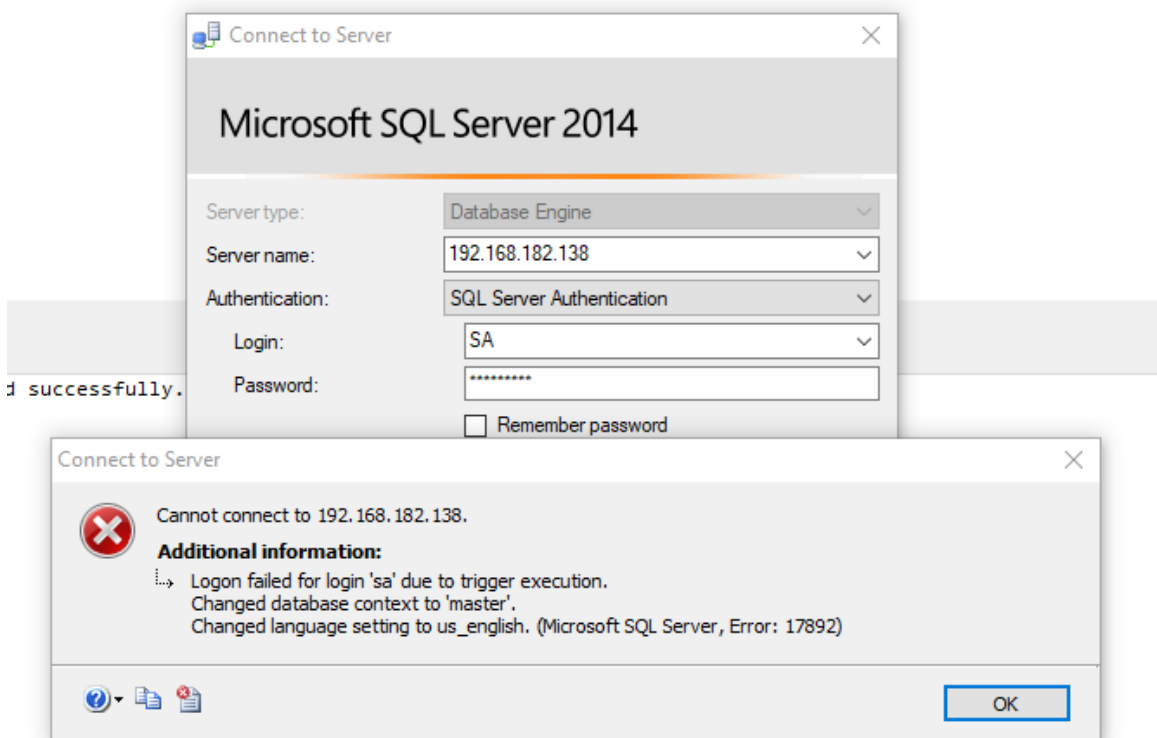
The trigger we created can be seen in the Server Objects part like this:



Now we see that the IP address is written into the blacklist when we attack.



Moreover, even if you enter the right password, access is denied.



I hope that this article has been informative.

Take care...

Malware Programming and Lockdown Virus

There are many different motivations for virus developers. Some of them develop viruses for cyber vandalism, earning money, damaging a company or taking revenge from somebody. Some, on the other hand, roll up their sleeves to this issue for developing antiviruses or for understanding how to create a virus, or how they work to take the necessary countermeasures.

Whatever reason leads these people to develop viruses, there is a problem that while you are doing researches for this issue, you encounter with fancy advertisements or phishing sites which pretend to be useful.

When I was a university student, I was into Java programming in my spare times. One day, I developed software which I also designed as a media player that acts like a virus because of its small error on the GUI. Multiple windows opened over and over and locked the computer like a Fork Bomb virus.

That day, I realized that there is no need, fund of knowledge or professional education to develop malware. The only thing you need is thinking simple! For instance, using `Java.awt.Robot` class with an infinite loop, you can fix the mouse cursor to 0,0 point of the screen. And if you run this software in the background, it can be used as a malware. Another example is creating software which copies itself to Windows Start Up while booting the computer, which creates new files and folders simultaneously and writes random bytes for bloating up the free storage area of the disk.

In this article, I am going to explain the steps to starting and following the right path for creating a virus. We will be covering a virus I have named as Lockdown.

Choosing the Ideal Programming Language

All programming languages have advantages and disadvantages, so we can't say a single programming language is the best for creating a virus. At this point, it is important to decide the programming language depending on what kind of a virus you plan to develop. Most especially the programming languages I recommend are Object Oriented, high-level languages such as C# and Java. If you use basic scripting languages, the features of your virus will be limited and because those languages can't be compiled, the source code will be open which is also a great problem. To run a virus created with a scripting language such as Perl and Python, an interpreter is needed to be installed on the victim's computer.

Another important factor is, in which language you are good at. For example, if you can't make a decision between C# and Java, and let's say that you are better at Java, you can easily choose it since both languages are Object Oriented. My personal choice has always been Java. This is also an advantage because it is relatively harder for anti-viruses to scan JAR files. Because Java is supported by all popular operating systems, it will run on all platforms apart from Windows. Unless you add a feature to make it only work in the Windows operating system.

Start-up Stage

First, you have to decide the features of the virus that you want to develop. In our example, the virus should copy itself to the computer and lock it continuously. What can be done for the virus to lock the computer? An encryptor to encrypt the files or a virus which will

move the files to a hidden folder as the famous Wanna-Cry does? This is totally up to your imagination. In my example, my preferred method is thinking simple and creating something which prevents the user from using the computer and in order to recover it, something that requires a password like an encryptor.

We can make the virus to copy itself to the Windows Startup folder. That way, the virus will be run each time the computer is turned on. Another method would be letting the virus copy itself to random folders and create a Registry Key. But in this article, my preferred method is the first one. As a bonus feature, we might want to hide the metadata of the virus. Metadata contains details such as the infection time of the virus. Therefore, it will be a good surprise for the victims if the virus changes the creation, modification and last access when it runs for the first time.

About the coding of the virus.. From now I will be explaining the functions with the codes, so you can download the source code of our example virus from the link below. That way you can follow the steps easily.

Download Link

<https://drive.google.com/file/d/16DNydtUz91sDP53S-dOQNk3waZixRNR8e/view?usp=sharing>

You even can compile and test the source code, but you will have to use your own image files instead of the ones I have used for the virus.

MetaStealth()

This function is for hiding the metadata. You can use FileTime for a specific date, but our example function sets all 3 metadata information to 1970.

About the other functions;

makeadminaccount()

This function executes terminal commands by using the Java process and runtime classes. This function also creates a new user with admin permissions and deletes the process log records. For doing all this work, the virus should be running with admin permissions.

copytostartup()

In this function, the virus first checks its own location and if it is not already in the StartUp folder, it decides to copy itself to the specified folder. After it copies itself, whenever the computer starts, the virus will also trigger itself. But as you can see, the copied virus is a JAR file, not an EXE.

If the file was an EXE which contains admin manifest, it would request admin permissions as it did before while being infected. Thereby, the user would be aware of the issue and deny it. So, the virus would not be executed. Briefly, the virus will finish the processes that need to have admin permissions when it is first copied, so it won't make any administrative requests. That way, as a bonus we will make the antivirus software harder to detect our virus.

getSHA256()

The aim of this function is to calculate the SHA256 table of a given string value. By this way, the software will compare the table to the SHA256 table instead of the password entered to the textbox by the victim.

So why did we store the password that way? If we stored the correct password as a string in the program, anybody who decompiles the software would get the password. But now the only thing that he may see is the SHA256 hash of the password and that will be useless for the decompiler.

Lock()

We can say that this is the main function of the virus. The *requesting password after locking the computer and unlocking it if the correct password is entered* processes will take place in this function.

As you can see in the code, we first calculate the screen resolution and then create a new iFrame which has the same size as the screen. So we can cover the whole screen and make the computer unserviceable. That way, we also get rid of the minimize button. After that, we set the JFrame utmost using the setAlwaysOnTop method in the GUI layer matrix. This way, we prevented other programs and the desktop from being seen with Windows and TAB keys.

Now we create the password input screen over this black panel. When the user enters the password and clicks on the unlock button, if the password is wrong, we display the cookie monster on the screen and if it is correct, we set our action listeners and start a daemon thread to protect the virus while running. Now, let's talk about the context of the daemon thread.

DaemonThread:

When the virus is running, the victim will try to fight against the virus instead of trying to enter the password. For such a situation we have to think about the actions to be taken against the virus and take the countermeasures to protect the virus by itself.

So what can the potential moves be? For instance, the victim can start a process from the task manager, which can stop the virus working or end the relative process. Or he may use CMD (Windows Command-line) and take the same actions. There are some methods that can be used to prevent the process of our virus from not being ended. For example, we can create a second virus process and let both processes check each other. That way, if one of the processes ended, the other can start it once again. If the victim cannot access the CMD or Task Manager, he/she already cannot take these actions and there will not be any problem for our virus.

When the virus runs, this daemon thread will also run with it every 300 milliseconds, it will close the Task Manager, CMD and File Explorer. Thus, when the victim tries to open those tools, they will be automatically closed before they even come up to the screen. That means, our virus is safe for now.

Lastly, you may ask why daemon thread but not normal thread, I can say that daemon threads are better in GUI software on Java. Because they use fewer system resources than the normal threads and because it won't cause a bug to the Event Dispatch thread, our virus will work properly.

Packaging Stage

After completing the coding process of the virus and got the JAR output, our virus is almost ready. But we have something missing...

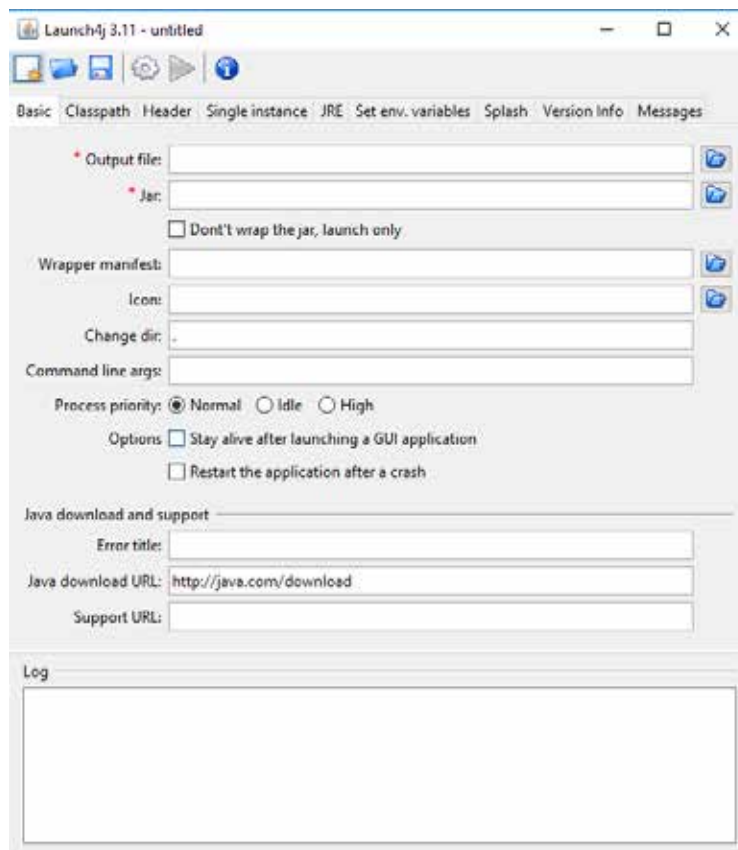
When the JAR file is running, because the virus is not working with admin permissions, copying itself to the StartUp folder and creating admin account features will not work. That means that when the victim restarts their computer, she/he can save himself from the virus. So how can we make the virus demand admin permissions?

At this point, EXE wrapping comes to play. What we need to do is converting the JAR file - namely the virus - into EXE and make it request admin permissions. We convert the file because only EXE files can request admin permissions. The reason for this is that the other file types are not executable. Our JAR file is like a command list that works on Java Virtual Machine. So the admin permissions can't be given to the JAR file itself, but to the javaw.exe.

For the wrapping process, you can search for different software on the internet. My choice is Launch4j.

Download Link: <http://launch4j.sourceforge.net>

You will see the below screen when you download and run the software:

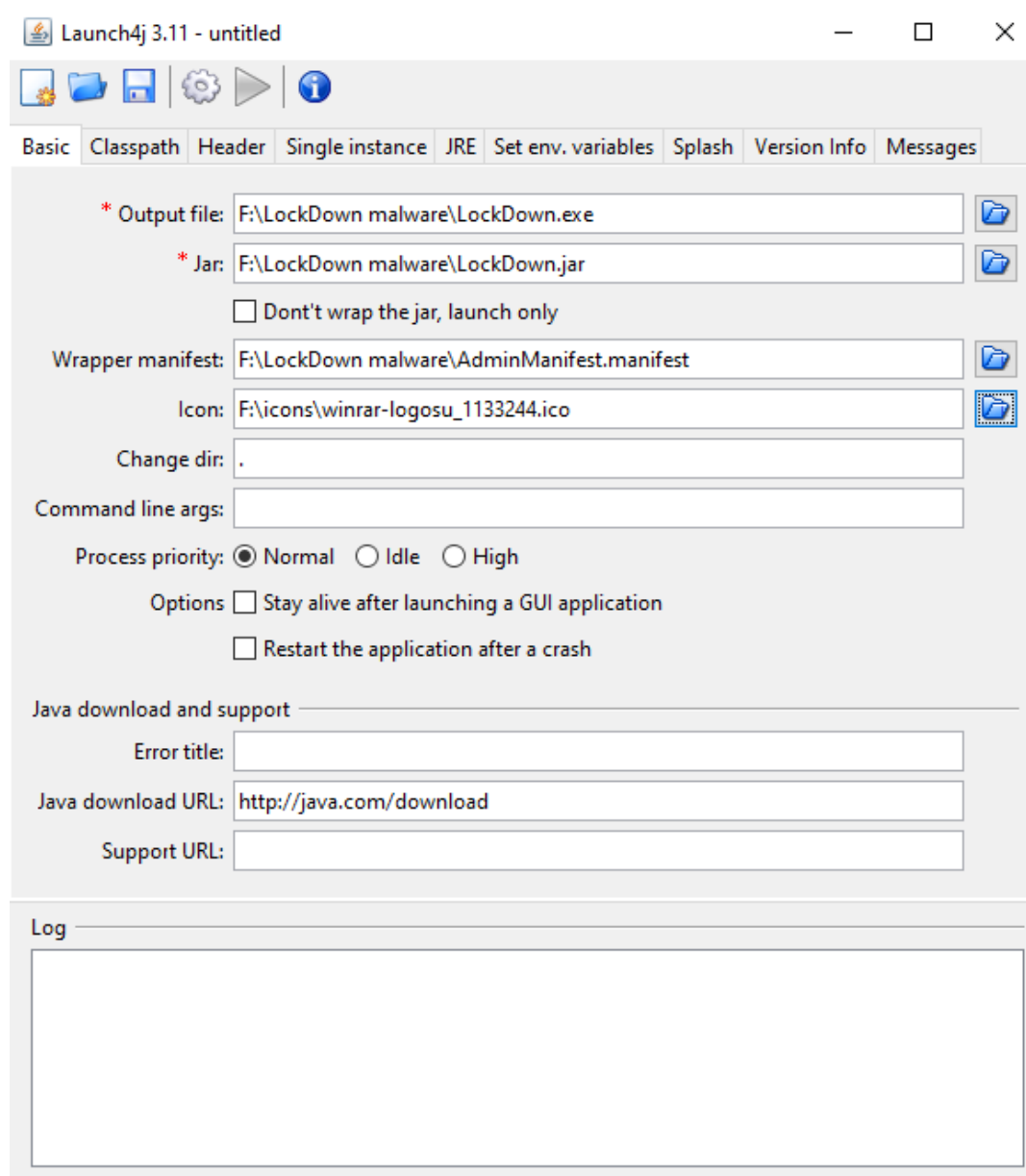


Here we have to enter the JAR file which we've exported and the wrapped EXE output addresses. Then we need the admin manifest file for the EXE file that the software will prepare to request admin permissions. You can find that file on the internet, but it is already included in the file that you've downloaded to get the source code of our virus.

You may even want to change the icon of the software.

For that, download the .ico file and use it with Launch4j. Last, To export the EXE, you have to enter to the JRE tab and enter the min and max JRE versions. You can set them to 1.00 and 8.00 and click on Build Wrapper (gear icon) to create the EXE file.

When you make the configuration, the program will look like below:



Now that we have the EXE file, our virus is ready to go!

When you run the virus, it will demand admin permissions, and if you accept that, it will first infect itself to the Windows StartUp folder and lock your computer as seen on the image below. To get rid of the virus, you have to enter the correct password. This will stop the virus from working and after you will have to delete it, or you can delete the virus by using a Live Linux distro.



As you can see, we created a virus similar to an encryptor.

Although it is an EXE file, the antivirus software and Windows defender cannot easily detect it because it actually is a wrapped JAR file.

The desktop cannot be reloaded in some computers because of a simple bug. In such a situation open the Task Manager and restart Explorer.exe.

Warning: The context of the article and the source code of the Lockdown virus is for educational purpose only. I hereby declare that any damage that is caused by the virus is not under my responsibility.

P.S.: If you run a compiled version of the virus accidentally, recovery password is “Adem1234”.

Network Programming with Scapy

Even though the simplified version of the packet structures on the network is constantly in our hands, we may have new discoveries when we examine these packages bit by bit. In order to listen to the desired packages through the network and see the details of the bits we want, we can use the visual interface part of the programs like Wireshark and the command line tools like tcpdump among the terminal user interfaces.

If we included the concepts we mentioned on the network to programming, it would be effortless for us to experiment, make changes as we wish, or examine the parts of our interest in depth. In fact, if a request did not go the way we want, we could take measures related to it. It would even make it possible to change the packages we listened to and put them on the line again. If we could use a language such as Python, interactively from the command line, or if we wanted to write the code into a file and pack it as we wanted, we could have brought it to a format useful in detecting many problems.

Scapy is a package manipulation library that can analyze and parse many protocols, including those mentioned, making it possible to manually generate packages, disassembling layers on the package, replacing them with other layers, and much more.

Speaking of such concepts, it is important not to forget that we are able to do the work of many programs such as Nmap, traceroute, tcpdump with Scapy. For example, we can determine different TTL values in the packets that we create and provide similar functions to traceroute functions according to the responses before certain amount of time runs out.

As a matter of fact, there are many different libraries written with Scapy. This includes Wi-Fi traffic injection or those which go to such high levels enough to make an HTTP request with Scapy.

When the tools used in cyber security do not provide the desired feature or there is no suitable tool to do the desired job, it usually takes a very short time to complete these deficiencies with Scapy and produce very understandable results.

Scapy can save packages in PCAP format. It can even read saved PCAP files or other programs by itself and make it possible to play with them by transferring them to their own data structures.

Scapy automatically performs checksum calculation, which can be an exhausting process, even though if done correctly, during package creation or when bits on the packages are changed or overlooked. When creating a package, you can work with Scapy to define the requested fields, often without any additional requirements.

Installation and Usage

Since it may take a lot longer to talk about Scapy, mentioning the areas of use, let's move forward hoping that I drew the necessary attention. In this section, we are going to talk about installing Scapy on the system. For this we will work on Ubuntu 16.04. Pre-installation and using some of common commands on the terminals of GNU/Linux systems shall make it easier for you. Finally, you will need an internet connection to implement the commands here.

Since many dependency needs can occur during installation, let's move forward by seeing the errors we may

encounter. We will proceed to learn how to correct these errors, even if we are addressed with too many error messages. Therefore, we will consciously write some commands incorrectly until the installation is complete and talk about what should be done.

We will install Scapy using Python 3 and use PIP for package installations. In addition, since we would not like to do the installation all over the whole system, we are going to install a virtual environment and install packets in this environment as much as we can. An advantage of doing so is that you will find the chance to work with two different versions of Scapy. Besides, this also shall enable us to create a medium where you will question the project-specific dependencies. The virtual environment and the files we will run will be the directory called **workshop** created under the home directory. Let's move forward by creating it and setting up Pip and virtualenv:

```
sudo apt update
sudo apt install python3-pip
pip3 install virtualenv
pip3 install --upgrade pip
```

```
mkdir workshop
cd workshop
```

The commands are: setting pip for Python3 up, and then installing the virtual environment package using pip, and updating of Pip using itself, respectively. Finally, we create and open the directory called workshop. Now we create and activate a new virtual Python environment (virtual environment) in workshop:

```
virtualenv -p python3 venv
source venv/bin/activate
```

To verify it, you can check to see if *venv* is written at the beginning of the command line.

This statement will be useful to check which virtual environment you are working on if the name of the virtual environment you are creating is different from the name of the environment you are working on :

```
which pip # /home/$USER/workshop/venv/bin/pip
```

If you typed the commands in your home directory from the beginning, the output of the command will be similar to the one given above. Only your username would be written instead of *\$USER*. After verifying that Pip is on the project-specific virtual environment, let's install Scapy:

```
pip install scapy
```

With this command, Scapy will be installed on the system. Let's try running it now:

Scapy

If you pay attention to the output, Scapy opens with a lot of warnings are given in between. So are these warnings critical? Although the answer to this question depends on the things you want to do, the fact that there are no modules such as *ipython* or *cryptography* increases the chances of having problems. Therefore, we will try to move forward by establishing as many dependencies as possible. Now, to exit, let's type `exit()` or use CTRL+D, then install the missing packages using pip:

```
pip install matplotlib pyx cryptography ipython
```

If we try to run Scapy again after completing the installation, we can see a warning like `texlive` or ... in the output. While we're at it, let's load it. We will install `texlive` with `apt` :

```
sudo apt install texlive
```

Note: Of course, we have to get out of Scapy before we write this command.

After all this effort, let's open it up to enjoy Scapy :

```
scapy
```

When working with Scapy, we can use the `conf` command to verify which settings are currently running and also to test the setup we have done in simple terms:

```
>>> conf
```

One of the most important areas to be considered in this output is the section in which `iface` is written section. It is useful to edit it if the connection interface we want to use is not selected. For example, while wanting to work on an Ethernet card, you may see the interface for the wireless internet connection in this section; to edit this :

```
>>> conf.iface = "enp0s3"
```

In this command, `enp0s3` is actually the interface name, so it is useful to edit it according to your system and purpose (for example, your system can have values like `eth0`, `wlan0`).

Let's try to listen to a packet while we're all set up. Since listening is often referred to as sniffing, the name of the function that makes it on the Scapy is called `sniff`. Now let's try to listen to a package by keeping it simple :

```
>>> sniff(1)
```

If you want to write and run the command, you must encounter an error message like "Operation not permitted" (although most of the time it is not recommended, you are expected to not receive this error if you are working as a root user). To solve this problem, you need to open Scapy with a user with administrator privileges. Let's remember that this package is based on the virtual environment inside the `workshop` directory and we want to open this environment with `root`. After leaving Scapy, let's run the scripts that are prepared to simplify the installation:

```
sudo su -  
cd /home/$USER/workshop/  
source venv/bin/activate  
scapy  
>>> sniff(1)
```

In the first line, you are switching to root, and you can use your own username instead of `$USER` in the second line. In the next lines, there is a small example of the expressions that we took a look at so far. In the last line, we want to

catch 1 packet with a sniff and we leave it to listen. If you have active network traffic, this line runs so fast that it may be hard to notice it takes time. If your network traffic is stagnant, the program will wait on this line until a packet is gone.

If you are using Ubuntu on a virtual machine while your sniff () function is running and your network traffic does not have an incoming-out packet, you can try to open a new terminal and ping an IP address. It will appear here when the first ping request is gone. You should get a printout similar to this:

```
<Sniffed: TCP:0 UDP:0 ICMP:1 Other:0>
```

Optionally, you can also apply BPF filters during network monitoring. They will be similar to the ones in the tcp-dump command. For example:

```
>>> sniff(1, filter="tcp port 80")
```

After you receive the packages, you may want to assign it to a variable to examine:

```
>>> pkt = sniff(1)
```

```
>>> pkt
```

```
<Sniffed: TCP:0 UDP:0 ICMP:1 Other:0>
```

In order to capture more than one packet with sniffer, we can write down how many packets we want to capture instead of the number 1 in it. Let's play a little:

```
>>> pkt[0] # Details of the first packet in sniffed packet list
```

```
>>> pkt[0]/"HELLO" # Add "HELLO" as payload
```

In the first line, we get a summary about the first package in the PKT structure. In the second line, we add "hello" to the package as content.

It is possible to save packages in PCAP format and then open them with programs such as both Scapy and Wireshark.

```
wrpcap("filename", variable_name)
```

```
variable_name = rdpcap("filename")
```

```
sniff(offline="filename") # Directly save to a file
```

The wrpcap in the example is actually used as write pcap, and rdpcap as read pcap. In the last line, the function we write will be saved directly in the file that we named.

We can use the IP () class to generate an IP packet from scratch. In the previous example, as we add "HELLO" letters to a package, we can combine the network layers in this way:

```
>>> IP()
```

```
>>> IP(dst="IP.AD.RE.SI")
```

```
>>> pkt = IP(dst="IP.AD.RE.SI")/ICMP() # Ping packet
```

In the first line, we looked at how to create an empty IP packet, and in the second line how to give the target IP address, but we didn't assign it to a variable or something. In the last line, an ICMP packet is generated after determining the destination address of the structure we have set up. The default packet generated of type ICMP echo request, which is often referred to as Ping. If we want to send this package to the real network:

```
>>> send(pkt)
```

It will be suffice. Since there are a few other parts that we need to set up, the package will probably not reach its destination even if it is actually dropped into the network.

While being able to create each part of the packet manually, we can also specify the target IP address. For example, if we want to send a ping package from 172.17.0.2 to 172.17.0.3;

```
>>> send(IP(src="172.17.0.2", dst="172.17.0.3")/ICMP()/"hello",
iface="enp0s3")
```

The ability to determine the source IP allows us to use whichever we want when we have more than one IP address and this can be used when running some tests in network management. To track what this command does, write `tcpdump -vv` and watch the packets from a separate terminal or track them with an application like Wireshark. We can also specify how many jumps we can allow on the network by giving a TTL to IP packets:

```
>>> send(IP(src="172.17.0.2", dst="172.17.0.3",
TTL=10)/ICMP(type=13)/"hello", iface="enp0s3")
```

In this example, we have identified 13 as the ICMP type. Just in case that it may be the first time you encounter with this type, let's talk about what it does; it is used to ask the timestamp to the server over the network.

Before concluding the first part, let's talk about how we can send a request and get back a response. By using the initials of Send-Receive, it is possible to send a packet and wait for response(s). Now let's take a look at an example:

```
>>> pkt = IP(dst="172.17.0.2", ttl=10)/ICMP(type=8)
>>> sr(pkt) # Sends and receives the responses, Layer 3
```

The last line we wrote deals with packets in the third layer of the OSI model. An example can be like:

```
>>> sr(pkt)
```

```
Begin emission:
```

```
*Finished sending 1 packets.
```

```
Received 1 packets, got 1 answers, remaining 0 packets
```

```
(<Results: TCP:0 UDP:0 ICMP:1 Other:0>,
```

```
<Unanswered: TCP:0 UDP:0 ICMP:0 Other:0>)
```

Sent packets, incoming responses and missed packets can be seen in detail in this output. The output above corresponds to a ping (ICMP echo request) that has been responded to.

As in this example, SR1 gives a much simpler output if we are just waiting for an answer. The number 1 in the function name means that we expect only one answer. Example of usage and sample output:

```
>>> sr1(pkt) # Returns one answer, Layer 3
```

```
Begin emission:
```

```
*Finished sending 1 packets.
```

```
Received 1 packets, got 1 answers, remaining 0 packets
```

```
<IP version=4 ihl=5 tos=0x0 len=28 id=15563 flags= frag=0 ttl=64 proto=icmp  
chksum=0xe5f0 src=172.17.0.2 dst=172.17.0.1 options=[] |<ICMP type=echo-re-  
ply code=0 chksum=0xffff id=0x0 seq=0x0 |>>
```

In the next issue, we will examine how to produce packages in the second layer of the OSI model, how to display the details of packages at different levels of detail and to examine the packages more clearly, to make TCP connections starting with three-way handshake with Scapy, to develop port scanning application in a fast and practical way with Scapy and the filtering of received answers, and see a way to create packets with only desired flags being set. Besides, we will mention how to set a timeout and an auto-retry limit to the responses, how to create a DNS request, and perform some simple attacks that are related to cybersecurity in Scapy. We will also recommend some resource suggestions for those who want to get to know Scapy further.

For more examples about this article, please see to Scapy's documentation at <https://scapy.readthedocs.io/en/latest/>.

Siber Yıldız 2019 Writeup

Preparation Question 1: Do you want to work with our IT company? (Bilişim şirketimizde çalışmak ister misin ?)

When the page's source code is examined

```
<p class="copyright">
  &copy; Maxim Theme. All rights reserved.
  <div class="credits">
    <!--      !!! Yonetici Panelini tespit edebilecek misin?      -->
    <a href="https://bootstrapmade.com/">Free Bootstrap Themes</a> by BootstrapMade.com
  </div>
</p>
```

It wanted us to find the admin panel. I decided to run dirb, found the admin panel and claimed the flag:



Preparation Question 2: If you examine well you can find the answer. (Dikkatle incelersen cevabı bulabilirsin.)



When you take a look at the page's source code, it says that you need to look at another URL.



When we go to the URL, you see an algorithm structure we need to solve. The key (*‘anahtar’*) implies that we need to use *‘/’* to move in the directories and that the flag is in the fourth *‘/’*. When we examine the URL, as the algorithm explains, the part after the fourth key shall give us the flag. After hashing the expression *“87habythi15ng151.php”* with md5, we got the flag.

COMPETITION QUESTIONS

Question: This really is easy (Bu gerçekten kolay)

It made us download a file named *dikkatli.bak.dms*. When this file is opened with any text editor, we see the following expression:

```
flag: RDcxVWV3SUFPLM29RazBGV1Rnb0dsbURpWDJnckg1MÜgvZ09WFFZmbkFBWmhjczk3S291TGprSnZEMENtYmxnd2R6d0t0c2pKeH2KvW8zVHczSUtIRUB9PQ==
```

Question: Still warming up (Hala ısınmalardasın)

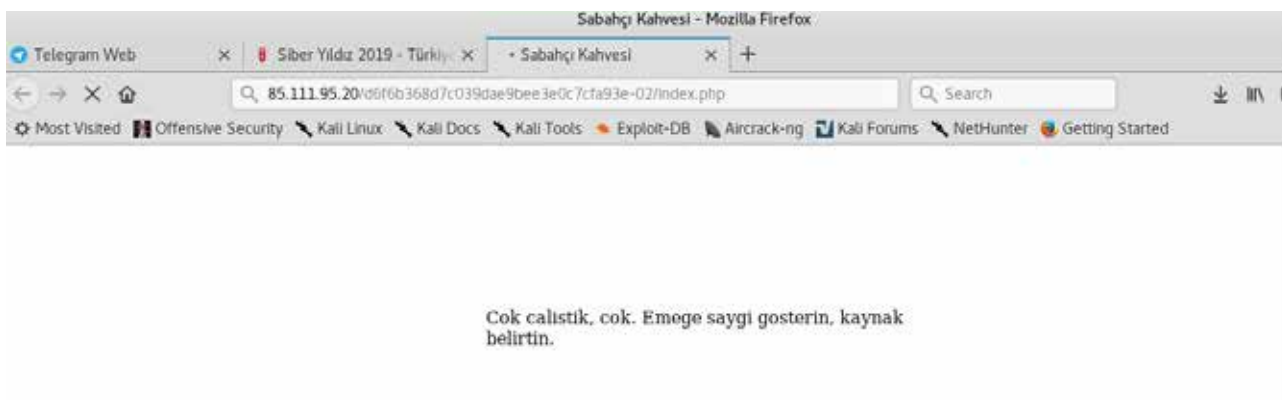


When we examine the source code of the question, we see some scripts. When decoded with the charcode decoder, the flag got extracted.

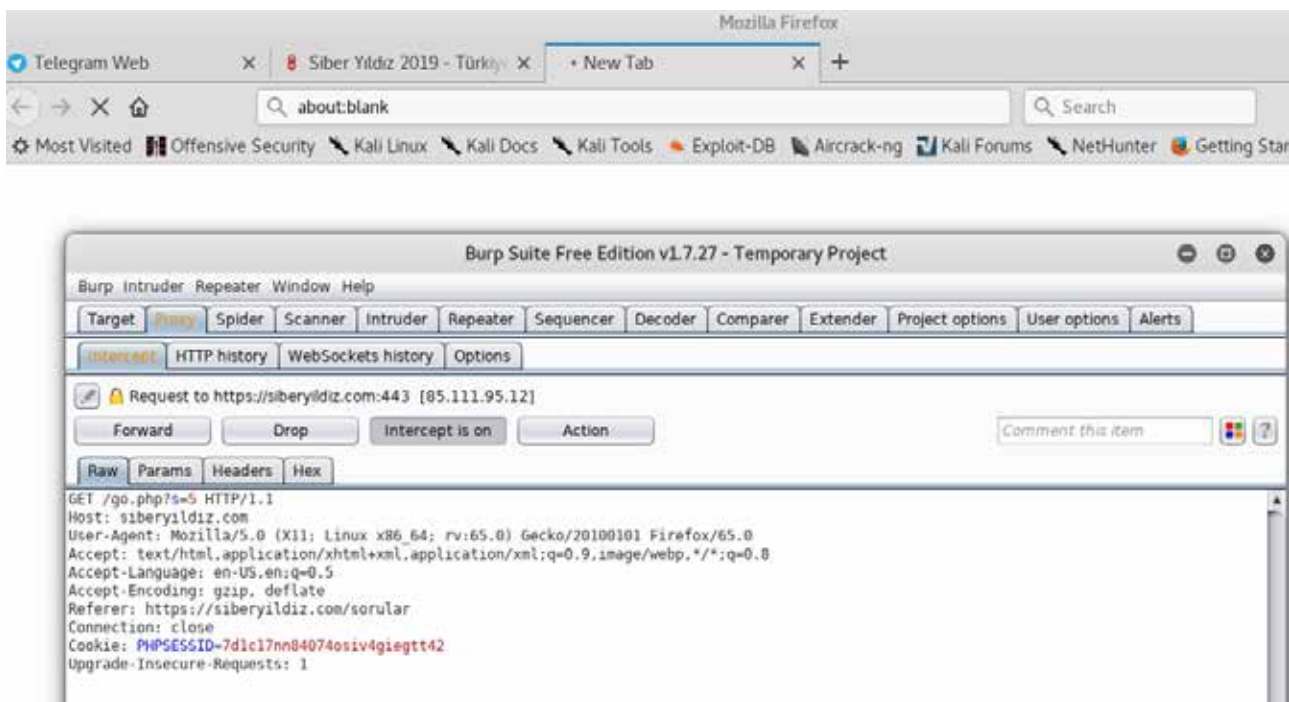
Question: Respect for labor (Emeğe Saygı)



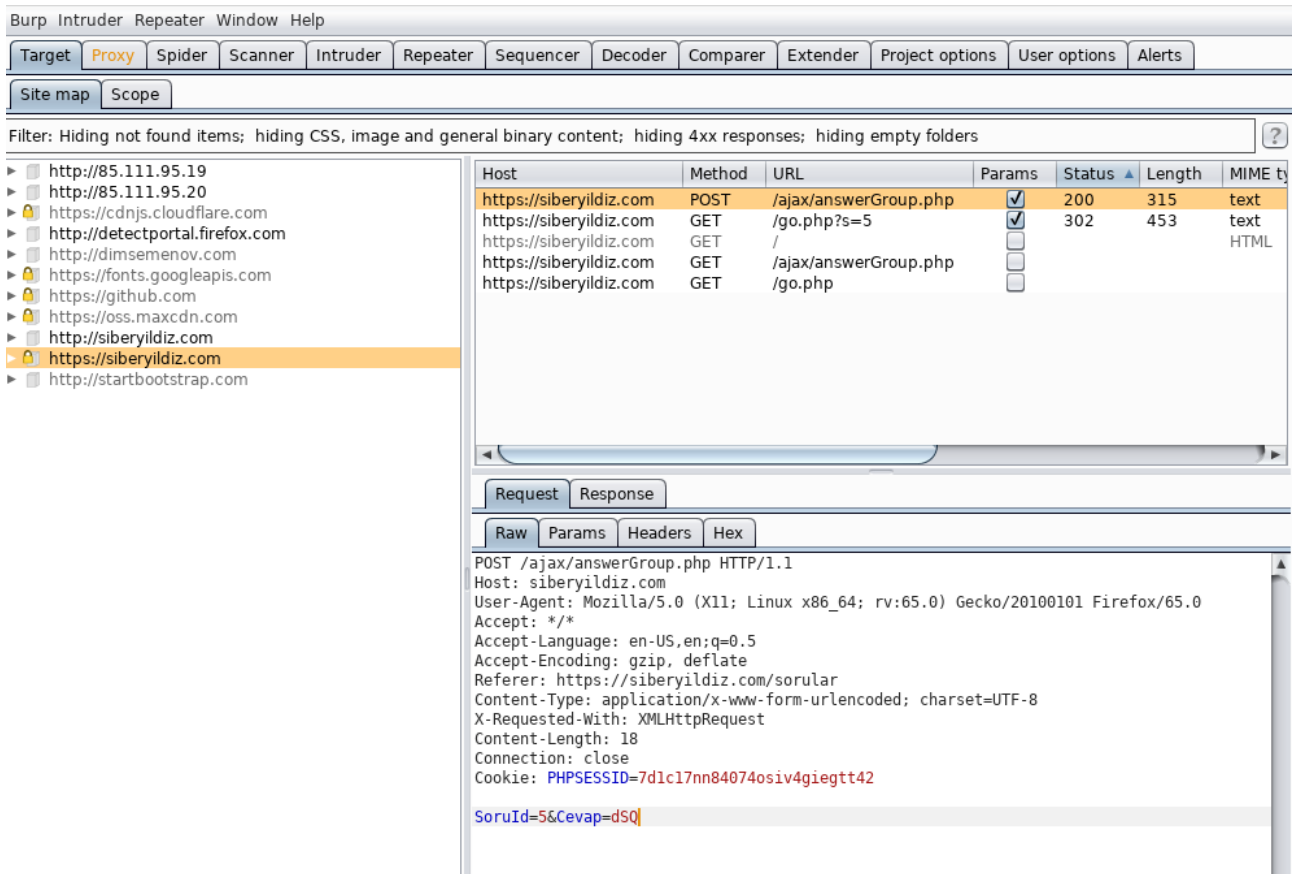
Answer:



When I opened the page, there was a sentence: “Cok calistik,cok. Emege saygi gosterin, kaynak belirtin.” (We worked so hard. Have some respect for labor and specify source)



So I decided to analyze with Burp; there was a cookie value as well as a Referer part. When I made random changes to the Referer part and sent the request, you can see “Kaynak belirtilere hediyeelerimizi gönderdik. Sağolun varolun.” (We gave prizes to those who specified the source. Thank you so much.) written in the response part. When we saw this part we thought that we were in the right place; only had to change the Referer part. So we tried various changes to the Referer. The “Sabah Kahvecisi” title of the website might as well have been a hint, so we googled that and found the *Sabahçı Kahvesi* song by Ferdi Tayfur and gave the YouTube link of it to the Referer but failed.



When I looked at the site map section of Burp, I saw a section called Cevap (Answer) by the QuestionId. But that was just unnecessary thrill :) What we had to do so was to prepare a file that records the requests that came with the sniffer, thinking that it made a request to the Referer. The flag came when we looked at the file.

[HTTP_REFERER] => işte ödülün (here's your flag) : aFZPL0hkSjhvaThneGdIdkFMcUd1UFZLO-GNhmGxGSG1lak1VbXIROFVGO

FdxVWI3bjFYTW10bVVFmK5ZdTVQV3daRSsxWFZsbmZWZ3dLOFMveHZxQnc9PQ==



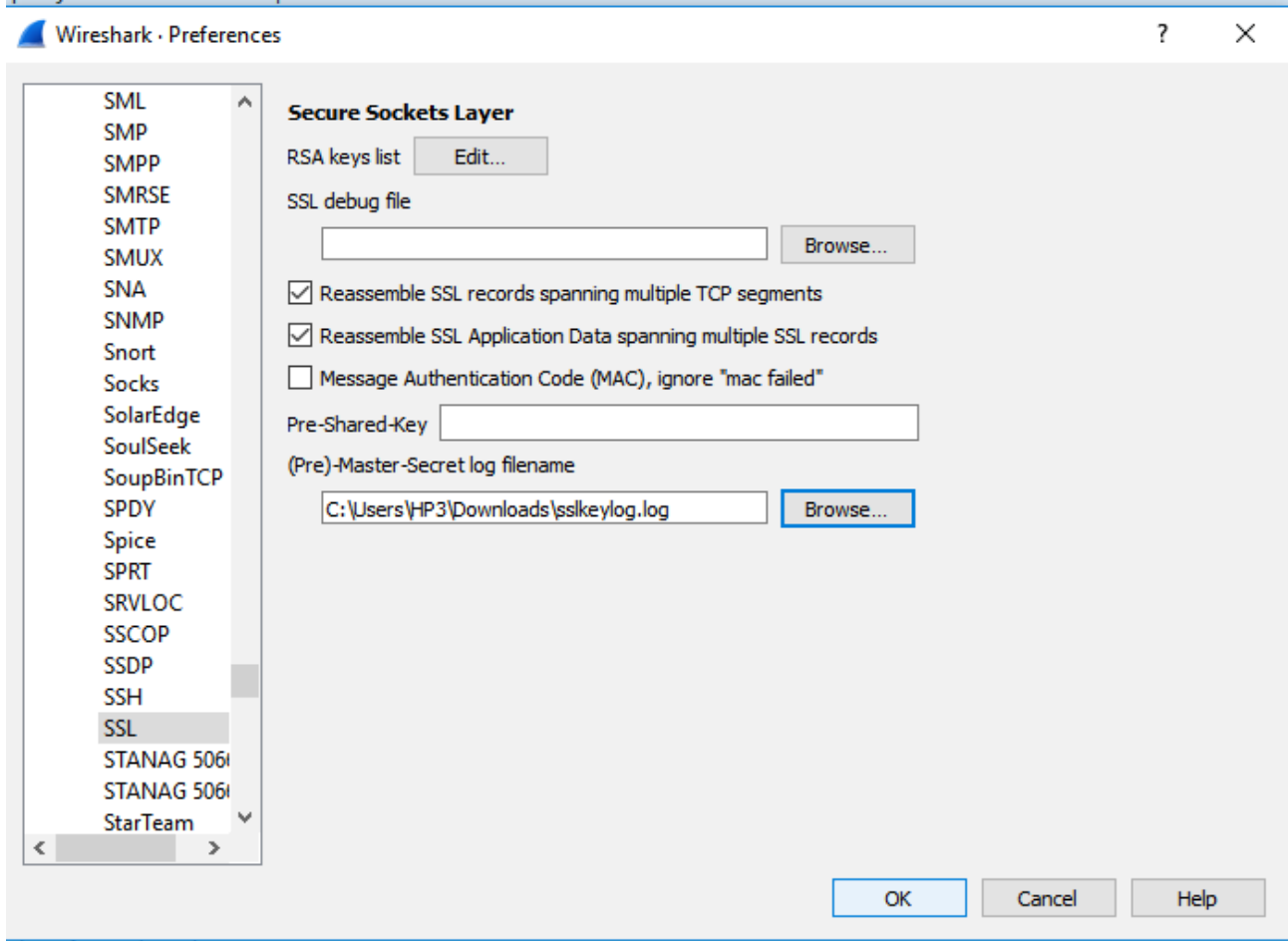
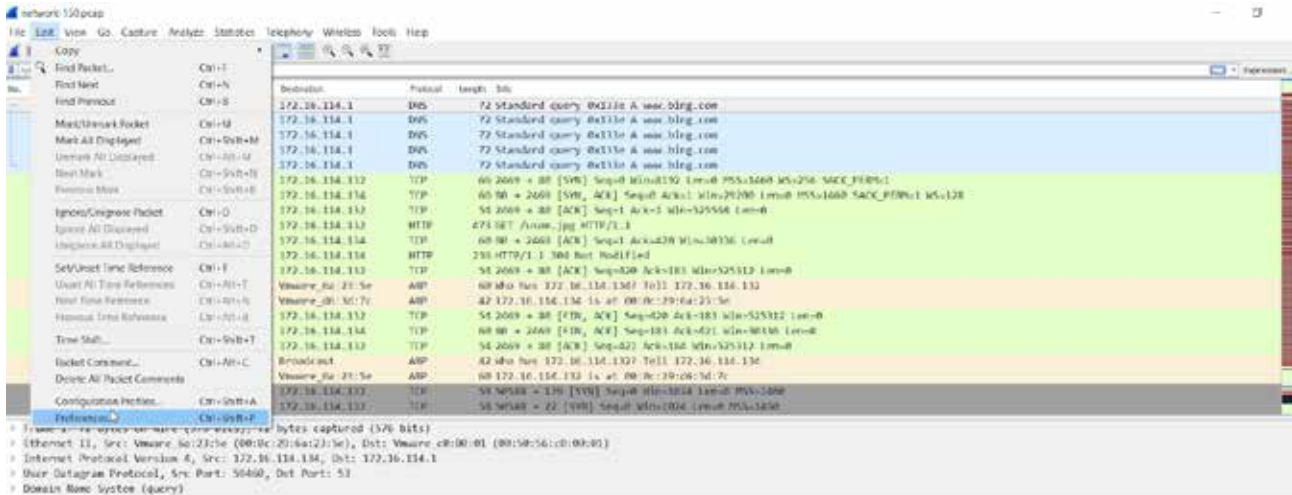
Question: Everything is virtual, the network is real.

As the start, we downloaded a wireshark file.

The screenshot shows the NetworkMiner interface. The top pane displays a list of network packets with columns for No., Time, Source, Destination, Protocol, Length, and Info. The bottom pane shows a table of exported files with columns for Pack #, Hostname, Content Type, Size, and Filename.

Pack #	Hostname	Content Type	Size	Filename
427		text/html	104 bytes	/
512	172.16.114.132	application/x-www-form-urlencoded	88 bytes	/
515	172.16.114.132		441 bytes	sdk
534	172.16.114.132	text/html	104 bytes	/
537	172.16.114.132	text/html	281 bytes	sdk
539		text/html	104 bytes	/
541	172.16.114.132	text/html	288 bytes	robots.txt
550	172.16.114.132	text/html	316 bytes	/
552	172.16.114.132	text/html	302 bytes	nmaplowercheck1505200568
556	172.16.114.132	text/html	104 bytes	/
559	172.16.114.132	text/html	316 bytes	/
568	172.16.114.132	text/html	287 bytes	HEAD
617	172.16.114.132	text/html	104 bytes	/
618			153 bytes	
619	172.16.114.132	text/html	316 bytes	/
621	172.16.114.132	text/html	283 bytes	HNAP1
629		text/html	282 bytes	
655	172.16.114.132	text/html	104 bytes	/
687	172.16.114.132	text/html	289 bytes	favicon.ico
853	172.16.114.132		223 bytes	sslkeylog.log
856	172.16.114.132	text/html	289 bytes	favicon.ico

Examining the package, it could be seen that there had been a few URL clicks and meaningless packages and nmap logs. When I exported the files within, there were again some senseless files except the sslkeylog.log which could be of use. In this way, even if we do not have a server key we can resolve the traffic passing over ssl.



After embedding sslkeylog.log in the SSL part of the Protocols section in Wireshark settings, the decrypted logs appeared in the Pcap, and when these were examined, it could be seen that there had been entries to some places.

798	69.609191	172.16.114.134	172.16.114.132	HTTP	444	GET /99037582138585721057129547823.pkt HTTP/1.1
799	69.610844	172.16.114.132	172.16.114.134	HTTP	339	HTTP/1.1 200 OK
800	69.666286	172.16.114.134	172.16.114.132	TCP	54	2499 → 443 [ACK] Seq=652 Ack=1914 Win=65280 Len=0
801	69.676105	172.16.114.134	172.16.114.132	HTTP	392	GET /favicon.ico HTTP/1.1
802	69.676812	172.16.114.132	172.16.114.134	HTTP	589	HTTP/1.1 404 Not Found (text/html)
803	69.728932	172.16.114.134	172.16.114.132	TCP	54	2499 → 443 [ACK] Seq=990 Ack=2449 Win=64768 Len=0
808	74.679636	172.16.114.134	172.16.114.132	TLSv1.2	85	Alert (Level: Warning, Description: Close Notify)
809	74.680029	172.16.114.134	172.16.114.132	TCP	54	2499 → 443 [FIN, ACK] Seq=1021 Ack=2449 Win=64768 Len=0
810	74.680418	172.16.114.132	172.16.114.134	TLSv1.2	85	Alert (Level: Warning, Description: Close Notify)
811	74.680475	172.16.114.134	172.16.114.132	TCP	54	2499 → 443 [RST, ACK] Seq=1022 Ack=2480 Win=0 Len=0
812	74.680557	172.16.114.132	172.16.114.134	TCP	60	443 → 2499 [FIN, ACK] Seq=2480 Ack=1022 Win=32512 Len=0

```

> Frame 798: 444 bytes on wire (3552 bits), 444 bytes captured (3552 bits)
> Ethernet II, Src: Vmware_6a:23:5e (00:0c:29:0a:23:5e), Dst: Vmware_d6:3d:7c (00:0c:29:d6:3d:7c)
> Internet Protocol Version 4, Src: 172.16.114.134, Dst: 172.16.114.132
> Transmission Control Protocol, Src Port: 2499, Dst Port: 443, Seq: 262, Ack: 1629, Len: 390
> Secure Sockets Layer
▼ Hypertext Transfer Protocol
  > GET /99037582138585721057129547823.pkt HTTP/1.1\r\n
    Host: 172.16.114.132\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:55.0) Gecko/20100101 Firefox/55.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate, br\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    \r\n
    [Full request URI: https://172.16.114.132/99037582138585721057129547823.pkt]
    [HTTP request 1/2]

```

```

GET /99037582138585721057129547823.pkt HTTP/1.1
Host: 172.16.114.132
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:55.0) Gecko/20100101 Firefox/55.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Upgrade-Insecure-Requests: 1

HTTP/1.1 200 OK
Date: Tue, 12 Sep 2017 03:32:08 GMT
Server: Apache/2.4.27 (Debian)
Last-Modified: Tue, 12 Sep 2017 03:30:18 GMT
ETag: "0-558f5a993e8bf"
Accept-Ranges: bytes
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive

GET /favicon.ico HTTP/1.1
Host: 172.16.114.132
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:55.0) Gecko/20100101 Firefox/55.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive

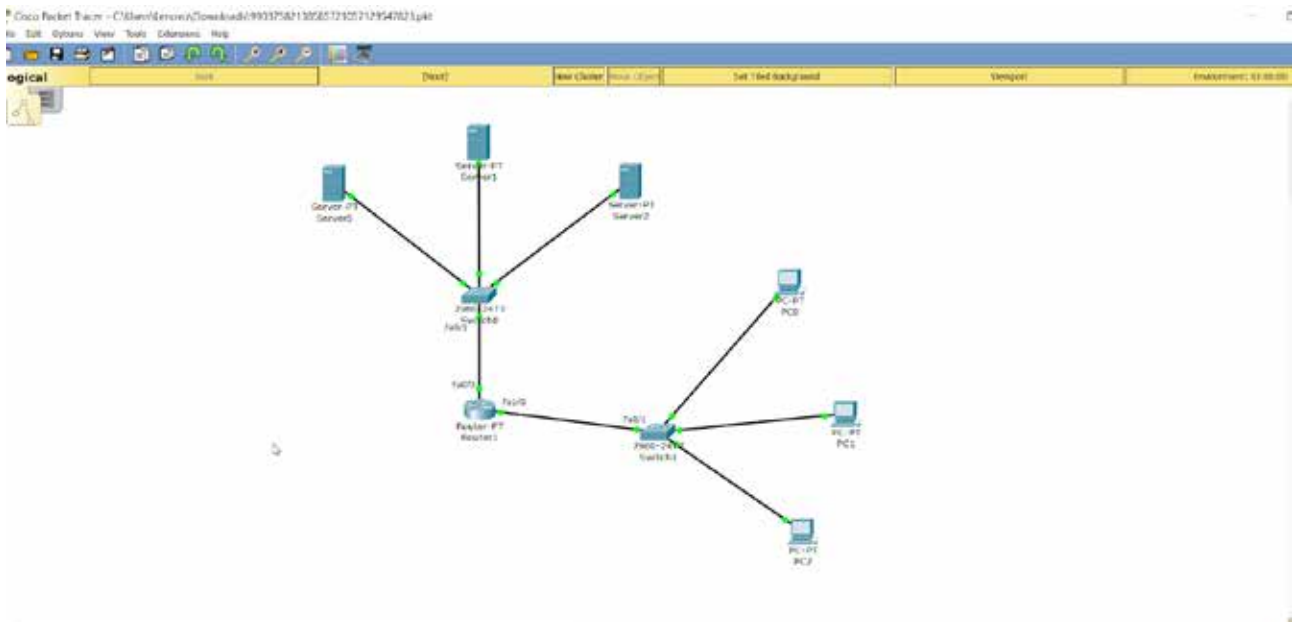
HTTP/1.1 404 Not Found
Date: Tue, 12 Sep 2017 03:32:08 GMT
Server: Apache/2.4.27 (Debian)
Content-Length: 290
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL /favicon.ico was not found on this server.</p>
<hr>
<address>Apache/2.4.27 (Debian) Server at 172.16.114.132 Port 443</address>
</body></html>

```

Http Stream Image

A file in pcap (99037582138585721057129547823.pkt) drew our attention thus we downloaded it from the server.



We have seen a programming language file called Dyalog. After some further research, we found a packet tracer file, downloaded the program and opened the 99037582138585721057129547823.pkt file.

After examining the router, we exported the *Startup config* file from the Config section.


```

|!
version 12.2
no service timestamps log datetime msec
no service timestamps debug datetime msec
no service password-encryption
!
hostname Router
!
no logging console
!
!
!
!
!
!
!
ip cef
no ipv6 cef
!
!
!
username usom password 7 0200085A0C5C022519061B49100317193C2439386D
!
!
!

```

Type 7 Password:

Plain text:

When we gave it to Cisco Password Cracker (Type 7), we got r0uterP@ss

Question: Password of OBELIX (OBURIX'in şifresi)

OBURIX'in şifresi

Toplam Puan:200 [Görevi Başla](#)

GÖREV KODU

KODU GÖNDER

Again, it made us download a pcap file.

No.	Time	Source	Destination	Protocol	Length	Info
141	23.169022	LgElectr_61:94:84 (- localhost ())	localhost ()	OBEX	26	Rcvd Connect
142	23.169078	localhost ()	LgElectr_61:94:84 (- localhost ())	OBEX	25	Sent Success
146	23.251235	LgElectr_61:94:84 (- localhost ())	localhost ()	OBEX	316	Rcvd OBEX fragment
149	23.276221	LgElectr_61:94:84 (- localhost ())	localhost ()	OBEX	316	Rcvd OBEX fragment
152	23.335079	LgElectr_61:94:84 (- localhost ())	localhost ()	OBEX	316	Rcvd OBEX fragment
156	23.392485	LgElectr_61:94:84 (- localhost ())	localhost ()	OBEX	316	Rcvd OBEX fragment
156	23.418651	LgElectr_61:94:84 (- localhost ())	localhost ()	OBEX	387	Rcvd Put continue "usom.png" (PNG)
184	27.153897	localhost ()	LgElectr_61:94:84 (- localhost ())	OBEX	16	Sent Continue
186	27.158941	LgElectr_61:94:84 (- localhost ())	localhost ()	OBEX	25	Rcvd Put final
187	27.159517	localhost ()	LgElectr_61:94:84 (- localhost ())	OBEX	16	Sent Success
189	27.287620	LgElectr_61:94:84 (- localhost ())	localhost ()	OBEX	22	Rcvd Disconnect
190	27.287765	localhost ()	LgElectr_61:94:84 (- localhost ())	OBEX	16	Sent Success

Arrival Time: Sep 13, 2017 08:27:43.132922000 EDT
 [Time shift for this packet: 0.000000000 seconds]
 Epoch Time: 1505105663.132922000 seconds
 [Time delta from previous captured frame: 0.009405600 seconds]
 [Time delta from previous displayed frame: 0.058795000 seconds]
 [Time since reference or first frame: 23.335029000 seconds]
 Frame Number: 152
 Frame Length: 316 bytes (2528 bits)
 Capture Length: 316 bytes (2528 bits)
 [Frame is marked: False]
 [Frame is ignored: False]
 Point-to-Point Direction: Received (1)
 [Protocols in frame: Bluetooth:HCI:HCI:BTLECap:BTLEConn:obex:data]

```

0000  02 00 11 27 01 dc 74 d8 00 00 00 46 0f 24 c1 d0  ...?..t..1'f.8...
0010  4a c9 a8 19 04 f3 bc 25 72 c8 7a e4 93 8f 7d f8  j.....%r'....:
0020  aa 04 a0 57 32 06 67 07 e3 04 30 f1 59 71 fb 8f  ...NRkqg...0.7q...
0030  48 84 ac 25 24 7a 08 7a 0a 0a a7 8a 00 6a 00 a0  ..r
  
```

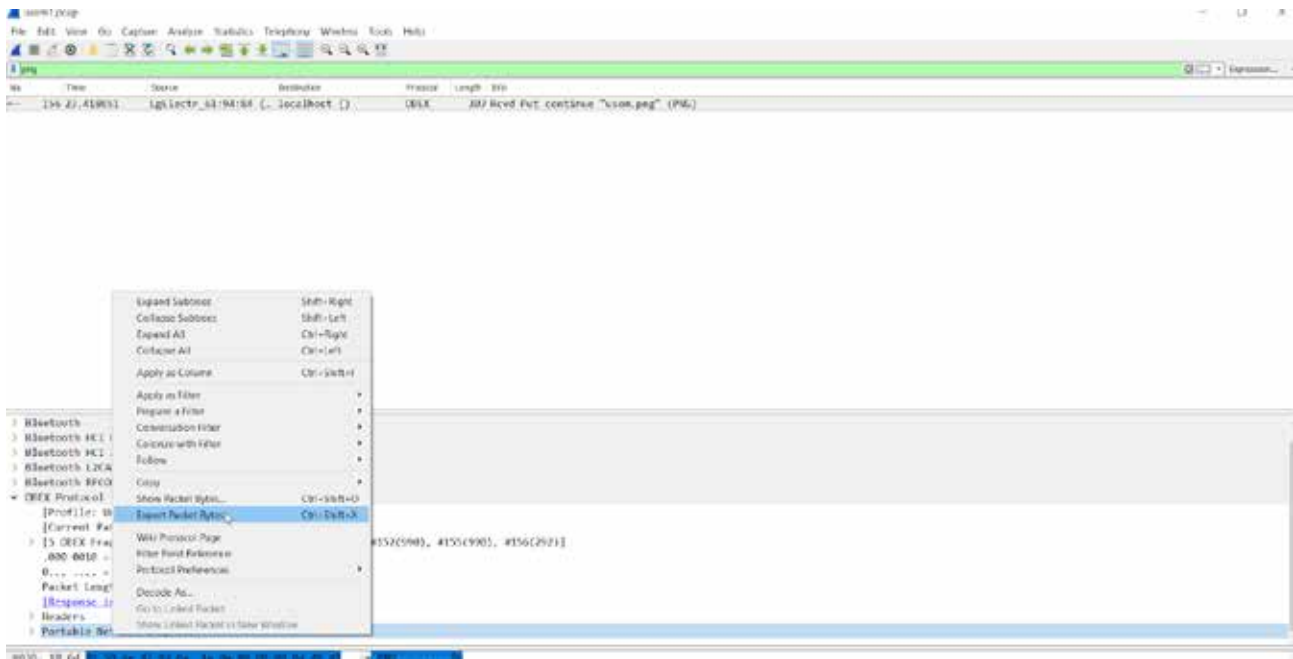
I reviewed the pcap and the word *obex* caught my eye due to its resemblance to the with the question; thus I started by writing *obex* into to filtering section.

usom1.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

png

No.	Time	Source	Destination	Protocol	Length	Info
---	156	23.418651	LgElectr_61:94:84 (- localhost ())	OBEX	387	Rcvd Put continue "usom.png" (PNG)



We see Bluetooth traffic after examining the pcap file. After investigating further, usom.png caught our eye so we tried to download with binwalk. However, the file got downloaded corrupt, so while looking for other download alternatives, we wrote png into the filter part and tried to download directly from the pcap file.

Right-clicking and downloading from the Export Packet Bytes option, the image told us the flag:

K0JwaTdxRUoydk9Ea3VpTW8rRVM1UT09

Question: There is a beholder, and a sought-after (Bir bakan var, bir de aranan)

At the start of the quest, we were faced with the following hash value: “0827206450376af3dce61d788dde-ba21f58dba35257fdb43c1872c096a36287f”



[LeoncioBecerraMacavei](#)

2019-02-01

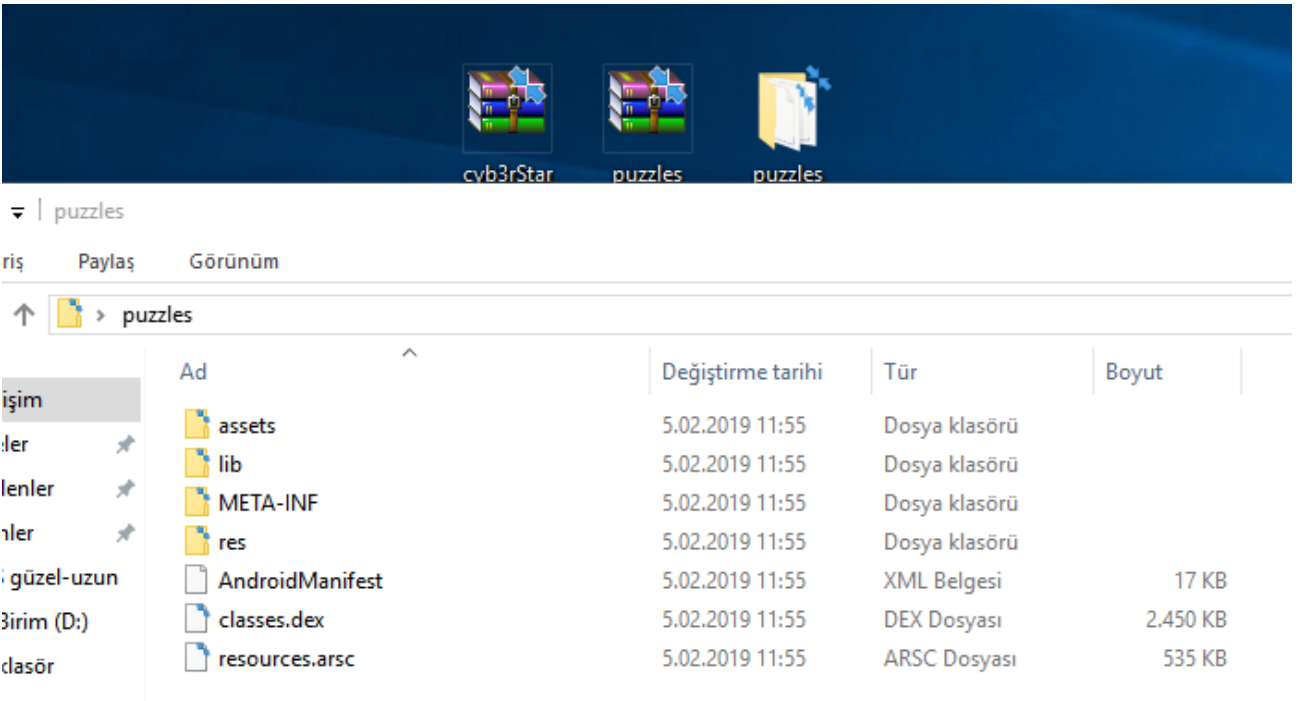
The malware tries to retrieve this file from a server:

<http://85.111.95.19/a5d8bccb8e1255fc72340eddab8be601-mobile01/cyb3rStar.zip> (Pass: Cyberstar_2018!.)

We saw a comment when we searched this hash value with VirusTotal:



When we downloaded and opened the zipped file, a file called puzzles.apk appeared.



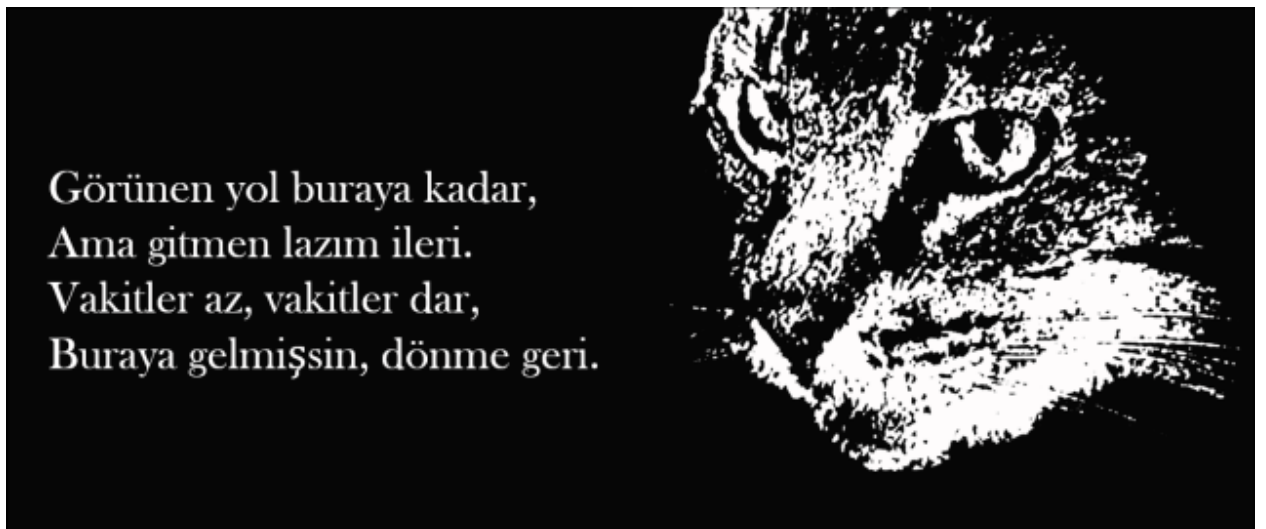
After changing the file's extension to .zip, we saw files inside, and inside /res/drawable, there were some image files.

DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	JPEG image data, JFIF standard 1.01
115805	0x1C45D	Zip archive data, at least v2.0 to extract, name: gizlidonya/
115878	0x1C4A6	Zip archive data, at least v2.0 to extract, uncompressed size: 7620, name: gizlidonya/thankyoucyberstar.gif
123023	0x1E08F	Zip archive data, at least v2.0 to extract, uncompressed size: 7620, name: gizlidonya/thankyoucyberstar_1.gif
130170	0x1FC7A	Zip archive data, at least v2.0 to extract, uncompressed size: 7620, name: gizlidonya/thankyoucyberstar_2.gif
137317	0x21865	Zip archive data, at least v2.0 to extract, uncompressed size: 7620, name: gizlidonya/thankyoucyberstar_3.gif
144464	0x23450	Zip archive data, at least v2.0 to extract, uncompressed size: 7620, name: gizlidonya/thankyoucyberstar_4.gif
151611	0x2503B	Zip archive data, at least v2.0 to extract, uncompressed size: 7620, name: gizlidonya/thankyoucyberstar_5.gif
158758	0x26C26	Zip archive data, at least v2.0 to extract, uncompressed size: 7620, name: qizlidonya/thankyoucyberstar 6.gif

When we looked through it with binwalk, there was the “thankyoucyberstar.gif” content that we see on most pictures.

DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	JPEG image data, JFIF standard 1.01
30	0x1E	TIFF image data, big-endian, offset of first image directory: 8
145579	0x238AB	Zip archive data, at least v1.0 to extract, name: gizlidonya/
145648	0x238F0	Zip archive data, encrypted at least v2.0 to extract, compressed size: 1342466, uncompressed size: 2100998, name: gizlidonya/hadibul.b64
1488383	0x16B5FF	End of Zip archive, footer length: 22

Yet, there was a zipped file in the ortakoy.jpg. When we tried to extract it, we saw that it was ciphered.



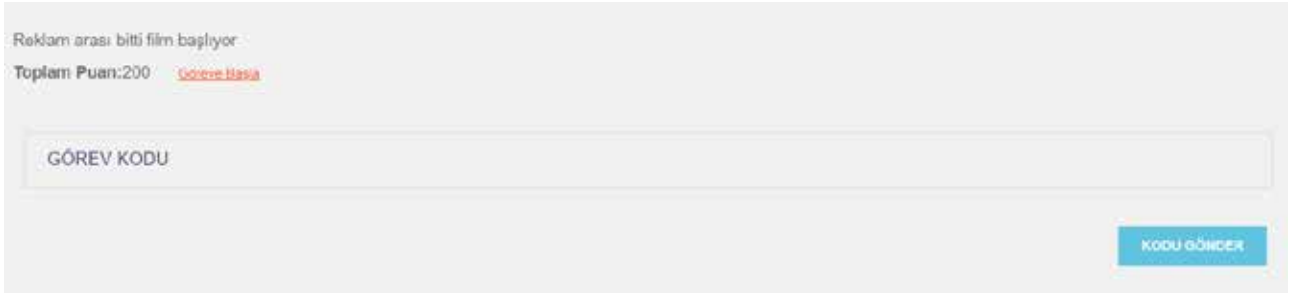
Meaning:

The path seen is only up to here,
But you need to go further.
Little is time,
You've come up to here, do not turn back.

```
IPTC Profile      : (Binary data 68 bytes,  
t)  
XMP Toolkit      : XMP Core 5.5.0  
Rights           : =?ufh]5%T7R2mGvQU?<n  
Image Size       : 1360x575  
Megapixels       : 0.782
```

The password of the zip file was found after looking into *sairnedemis.png* with ExifTool. When we opened the file, we found the *hadibul.b64* file and the flag was found when we ran the apk file.

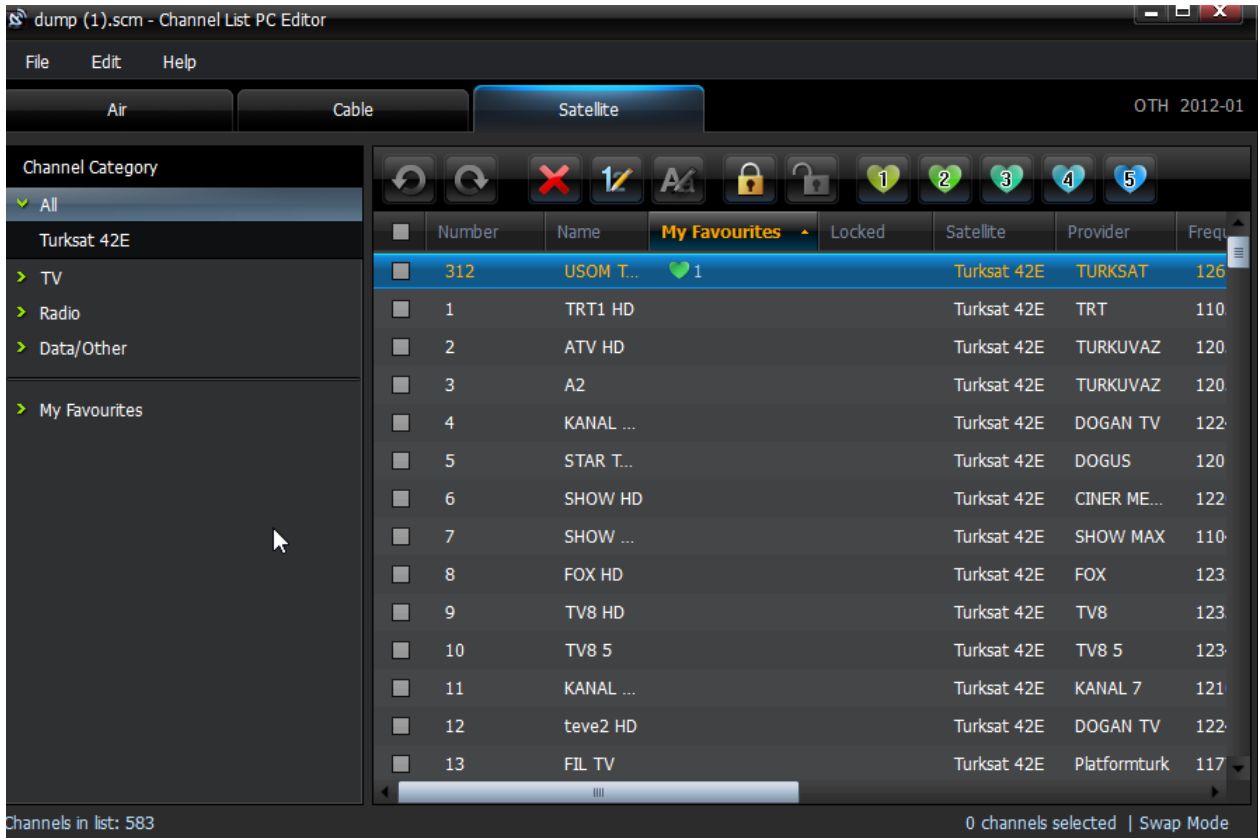
Question: Ad breaks have ended, the movie's starting. (Reklam arası bitti film başlıyor.)



Reklam arası bitti film başlıyor.

Toplam Puan:200 [Görev Başla](#)

We were given the *dump.scm* file at the beginning. To open the *.scm* file, when we did a search on google, we found and downloaded the appropriate editor and opened the file.

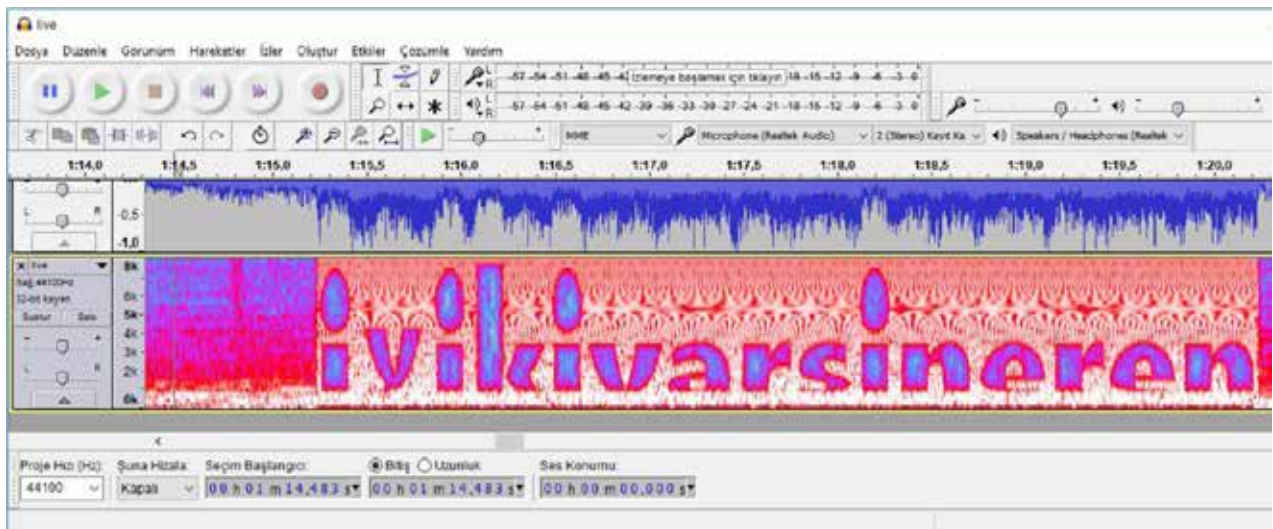


USOM TV came up. Opening this up with the editor, it gave us a scene from the Interstellar movie. While being so tired, it was so nice watching a scene from a movie I adore :)

There were morse codes on a scene, so we grabbed a paper and a pen and started to take notes, but that was a waste of effort.

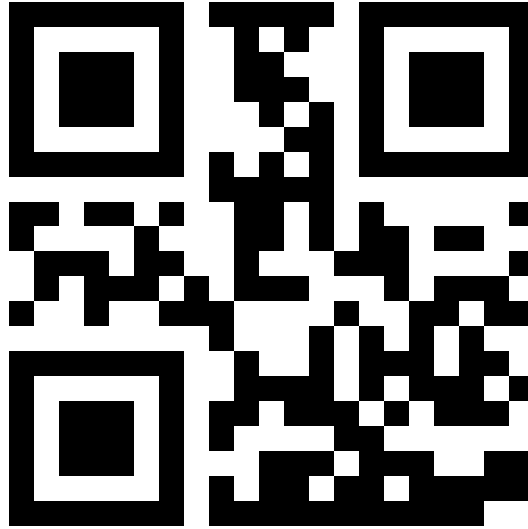
There was a sizzling corruption later in the video, so we decided to investigate the voice file using a powerful forensic tool namely Audacity.

In Audacity, we clicked *live* on the left, then *split* and started to examine the corrupt part. The flag was the md5 encoded version of *iyikivarsineren* - the result we obtained from the previous actions.



Question: QR Code Reader (Karekod Okuyucu)

There was a login screen on the page and a part where the users were listed. When we tried to take a look at the users, we got the “*sadece yöneticinin QR Kodu kabul edilir*” (only the QR code of the admin is allowed) message. Therefore I quickly generated a QR Code and embedded an SQL Injection as text, saved as png and it worked when I uploaded it to the website.



We got the following sentence:

```
1 admin 6b71dfdc4c5603272482f5b80db96a0a 5e14ce1f1fa3524ba07cb109549c594e
```

After decoding with md5, we saw that the password was admin1234567890. Logging in again as admin with this password, we got the flag.

Question: Do you like Kahramanmaraş ice cream? (Kahramanmaraş dondurması sever misin?)

P.S.: Kahramanmaraş ice cream is a renowned ice cream known especially by staying hard/rigid for a long time.

There was a game. When the game was over, we got directed to this URL:

```
yenibasliyor.php?r=6FAD329DF3870D30696C93460EBB7C29_498D3C6BFA033F6DC1BE4FC-  
C3C370AA7_
```

```
348DF46154717306D71E71C277E71082_
```

What came to our attention when playing the game is that we did not always have the control. If we tried to die intentionally, the game took over and made us not get killed. Thinking that there was something fishy with this, we took a look at the URL and the source code.

```

1 <!DOCTYPE html>
2 <html>
3 </html>
4 </head>
5 <meta charset=utf-8 />
6 <title>Oyun Zamanı</title>
7 <style>* { padding: 0; margin: 0; } canvas { background: #eee; display: block; margin: 0 auto; }</style>
8 </head>
9 </html>
10 </body>
11 </body>
12 <canvas id=myCanvas width=480 height=320></canvas>
13 <script type=javascript src=game/04.js></script>
14 </script>
15 var _0x4082 = [{"anytime": "anyup",
16 "mousemove": "mstopper",
17 "constructor": "copy",
18 "return (function() {",
19 "})<script>return (this) {",
20 "console": "warn",
21 "debug": "info",
22 "error": "exception",
23 "trace": "log",
24 "getelementById",
25 "myCanvas": "width",
26 "keyCode": "keydown",
27 "offsetLeft": "status",
28 "location",
29 "appendChild": "php?r=498d3c6bfa033f6dc1be4fcc3c370aa7_348df46154717306d71e71c277e71082_4",
30 "innerHTML": "498d3c6bfa033f6dc1be4fcc3c370aa7",
31 "fill": "clearPath",
32 "rect": "height",
33 "font": "loga Azial",
34 "fillText": "score",
35 "clearRect": "game over",
36 "reload",
37 "addEventListener"
38 }];
39 (function (_0x5729bc, _0x27cd87) {
40 var _0x2998d = function (_0x3548f7) {
41 while (--_0x2548f7) {
42 _0x5729bc["push"](_0x5729bc["shift"]());
43 }
44 }

```

There were so many hex expressions, all written in a script. When we converted from Hex to ASCII, we obtained a base64 encoded expression and after decoding it we were left with a text. Because it would be very difficult to convert them one by one, we wrote a script and by doing so, we got the source code.

Found : **baglanti**

(hash = 6fad329df3870d30696c93460ebb7c29)



Found : **son**

(hash = 498d3c6bfa033f6dc1be4fcc3c370aa7)



Found : **tekle**

(hash = 348df46154717306d71e71c277e71082)

We saw 3 md5 encoded texts in the URL section and decoded them. Later, we got:

yenibasliyor.php?r=6FAD329DF3870D30696C93460EBB7C29_

498D3C6BFA033F6DC1BE4FCC3C370AA7_348DF46154717306D71E71C277E71082_C4CA4238A0B-923820DCC509A6F75849B

When we added the md5 encoded version of 1 (C4CA4238A0B923820DCC509A6F75849B) at the end of the URL.

Mocking a little, they really were trying us to get an ice cream when we went to that URL :)

← → ↻ ⓘ Güvenli değil | 85.111.95.22/c98f16450d4754cd6fde30be9d0cfe84-mix03/yer

Bayrağı Kap

← → ↻ ⓘ Güvenli değil | 85.111.95.22/c98f16450d4754cd6fde30be9d0cfe84-mix03/simdibasladi1.php

Hadi bakalım.

← → ↻ ⓘ Güvenli değil | 85.111.95.22/c98f16450d4754cd6fde30be9d0cfe84-mix03/simdibasladi2.php

sonu yok bu gidisin

← → ↻ ⓘ Güvenli değil | 85.111.95.22/c98f16450d4754cd6fde30be9d0cfe84-mix03/simdibasladi3.php

bence vazgeç

← → ↻ ⓘ Güvenli değil | 85.111.95.22/c98f16450d4754cd6fde30be9d0cfe84-mix03/simdibasladi4.php

sonuna kadar ilerleyebilirim ?

← → ↻ ⓘ Güvenli değil | 85.111.95.22/c98f16450d4754cd6fde30be9d0cfe84-mix03/simdibasladi5.php

vakit varken bırak bu işleri

← → ↻ ⓘ Güvenli değil | 85.111.95.22/c98f16450d4754cd6fde30be9d0cfe84-mix03/simdibasladi6.php

benim limitim max url karakter sayısı kadar !

← → ↻ ⓘ Güvenli değil | 85.111.95.22/c98f16450d4754cd6fde30be9d0cfe84-mix03/simdibasladi7.php

pof, senle uğrasmak zaman kaybı.

← → ↻ ⓘ Güvenli değil | 85.111.95.22/c98f16450d4754cd6fde30be9d0cfe84-mix03/simdibasladi8.php

404 Not Found

← → ↻ ⓘ Güvenli değil | 85.111.95.22/c98f16450d4754cd6fde30be9d0cfe84-mix03/simdibasladi9.php

[Buradan devam et.](#)



← → ↻ ⓘ Güvenli değil | 85.111.95.22/c98f16450d4754cd6fde30be9d0cfe84-mix03/yenibasliyor.php?r=6FAD329I

Bayrağı Kap



[Başlamak için tıkla](#)

← → ↻ ⓘ Güvenli değil | 85.111.95.22/c98f16450d4754cd6fde30be9d0cfe84-mix03/gerçektenbasladi.php

Web Uygulamaları

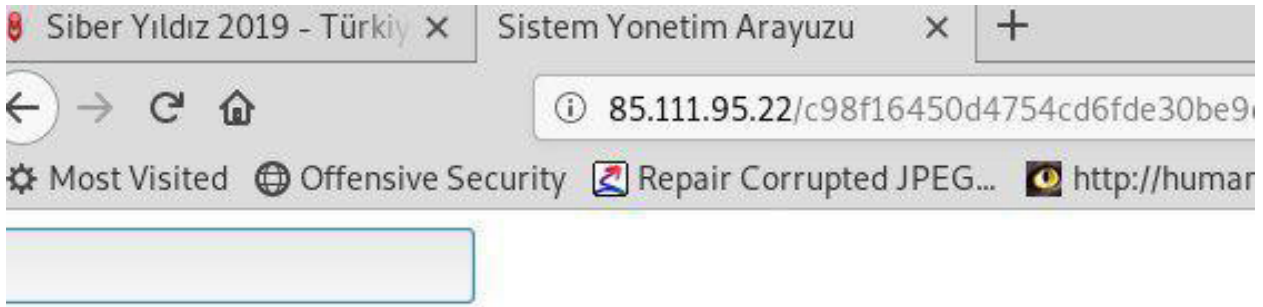
- [Gizli Portal](#)
- [Sistem Yonetim Arayuzu](#)

← → ↻ ⓘ Güvenli değil | 85.111.95.22/c98f16450d4754cd6fde30be9d0cfe84-mix03/bu_cok_gizli_bir_portal/index.php

bu portal sevdiğiniz dondurmalar hakkında bilgi vermek amaçlı kurulmuştur.

[Anket](#)
[İstatistik](#)
[Dondurma](#)

← → ↻ ⓘ Güvenli değil | 85.111.95.22/c98f16450d4754cd6fde30be9d0cfe84-mix03/akillidusun.php



```
flag
ozel/
systemd-private-d579b7af92374af49d86f4db27633112-systemd-timesyncd.servic
.X11
```

After wandering through the pages, we found somewhere we could execute commands.

There was a textbook, and we could run `ls`. Looking through with `ls /tmp`, we saw `flag` and `ozel/` (*special*) files. Just as we got excited, thinking we got so close, `cat /tmp/flag` did not make us read the file: only a response which said that the file could be read. We could not even execute `ls /tmp`. Unfortunately, without being able to finish this question, we ran out of time

A Security Guide for Your Android Device

The Android operating system, supported by Google; is an operating system used in many different devices, from mobile phones to televisions to tablets.

The instructions stated below may vary depending on the type of device used.

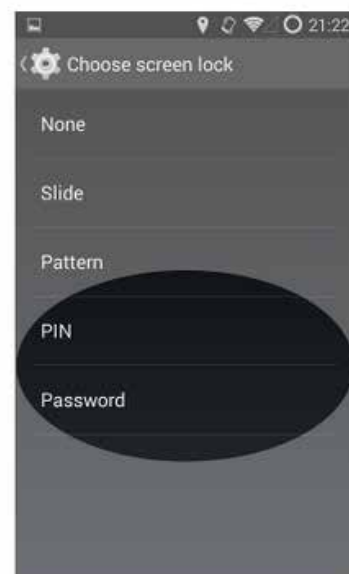
Minimize the data Google collects

On most Android devices, it is not mandatory for you to sign in to your Google account. It is possible to skip this option you'll see during installation. Yet, this may result in limited use of certain services. Besides, you can also edit your Google activity profile via <https://myactivity.google.com/myactivity>, determine which data is stored, or delete your activity data.

Set a PIN to your device

To protect your device, set a pin or an alphanumeric password.

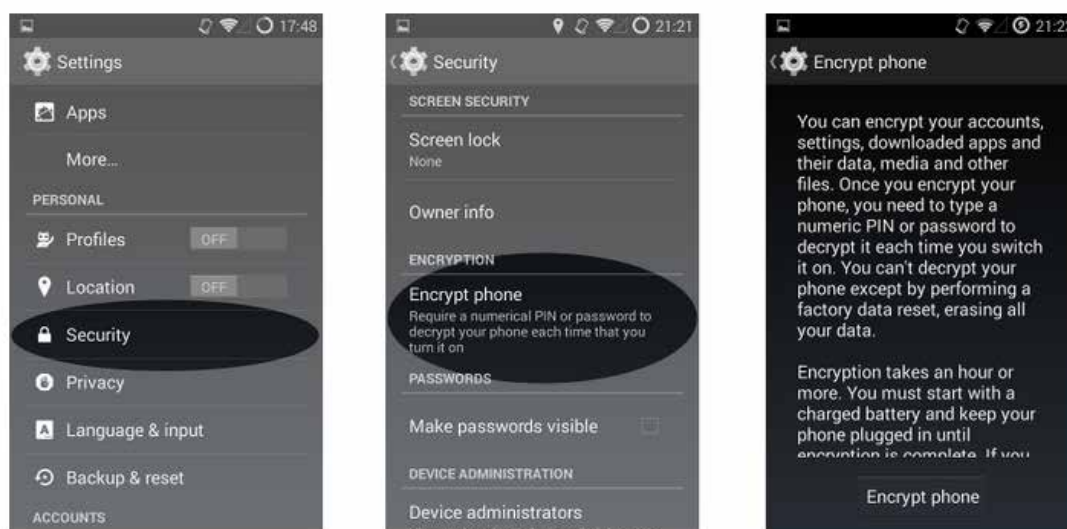
To set a PIN/Password go to Settings > Security > Screen lock



Encrypt Your Device to Protect Your Data

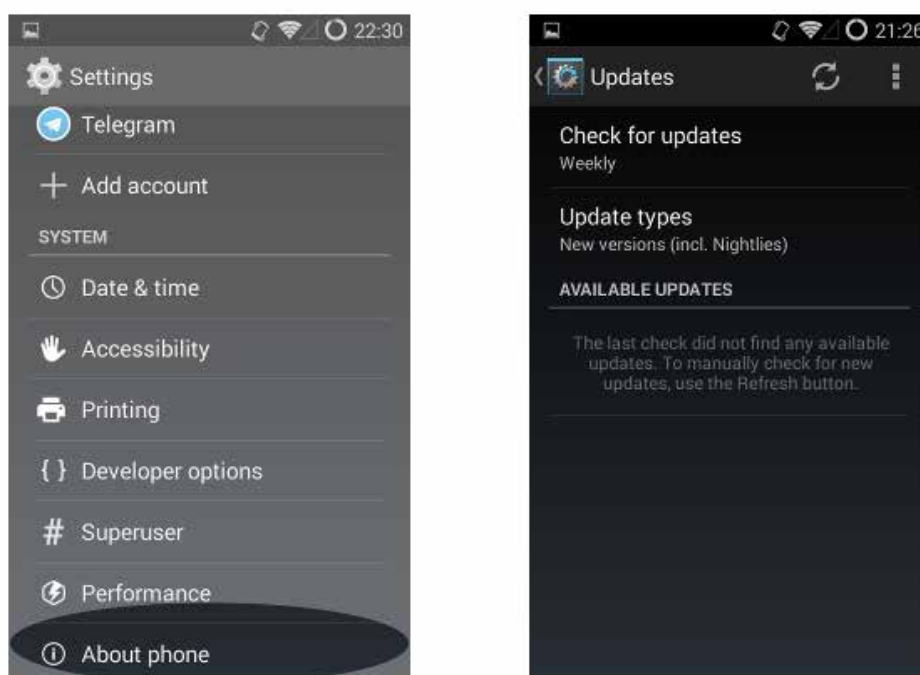
Differing from PIN/Password, you can encrypt the data inside your device with this setting. In order to do so, you first need to activate the PIN or Password and enter this information at each unlock. Since the encryption process wastes energy intensely, it is recommended you connect the device to the charger. In order to encrypt your device, you can also use the *Settings > Security > Encrypt phone/tablet* menu.

P.S. After encryption, it would not be possible to decrypt your data if you forget the PIN/Password. In this case, you can use the *return to factory settings* option but this will end up with all your data being deleted.



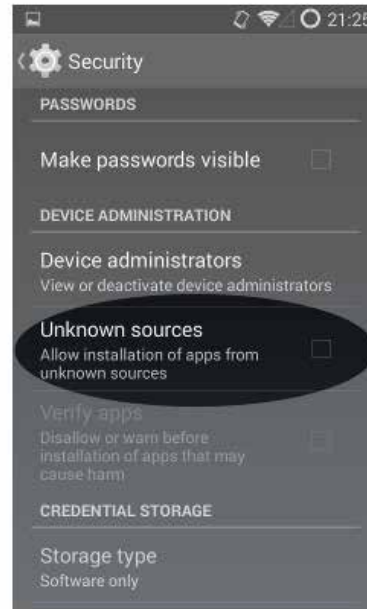
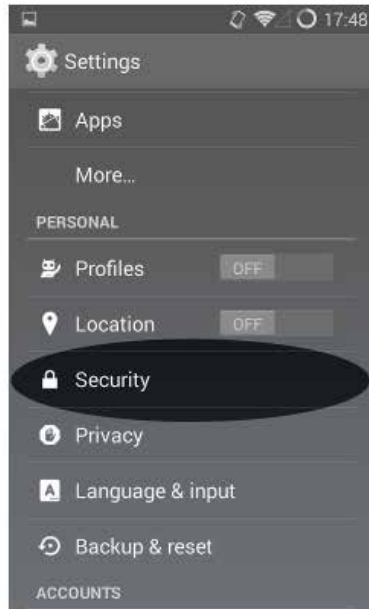
Keep your device and applications up to date

We also recommend you to keep all your devices' operating system and applications up to date, not only for Android devices. You can use the *Settings > About phone/tablet > System Update* menu settings for updating the device.



Do not download applications from untrusted sources

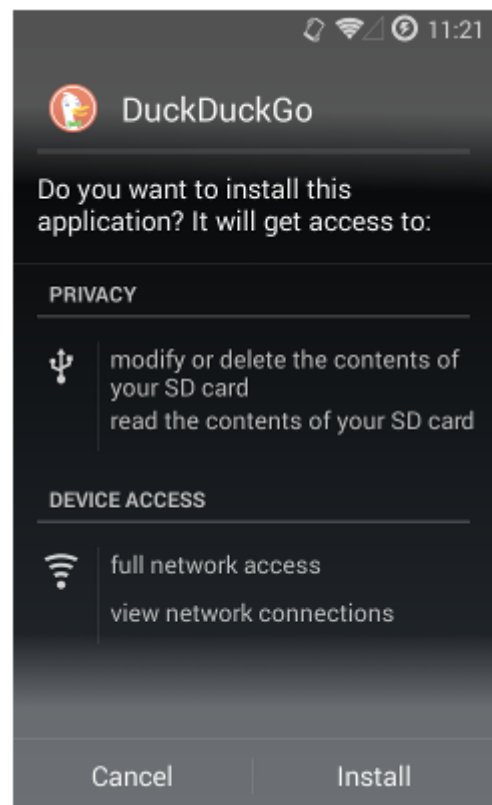
Do not download applications from untrusted, unknown sources. You can arrange the settings so that only applications which are from trusted sources are downloaded. To enable this feature, disable the setting at *Settings > Security > Unknown sources*.

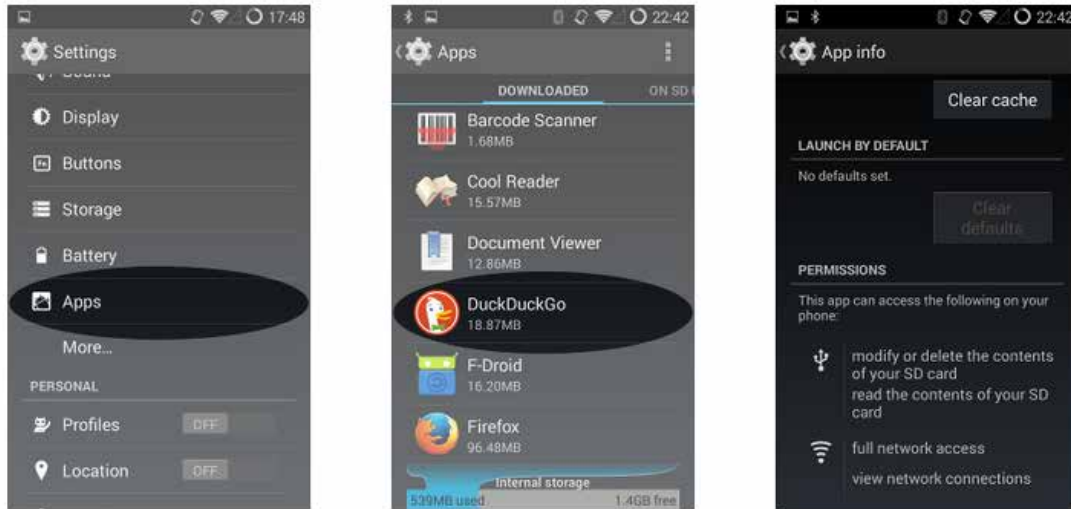


Review the application permissions

Definitely look over the permissions the application asks for during installation. Does the application require authorization other than its aim? For instance, does the application demand permissions to the microphone, camera, speakers or other sensors even though it has nothing to do with them?

Check the permissions of all applications including the old ones. If an app is granted excess permissions, question why and uninstall if necessary. Some apps may have requested extra permissions during an update, you may have missed some permissions or there may be an app you no longer use. You may change an app you already use for another which does the same thing but requires fewer permissions.





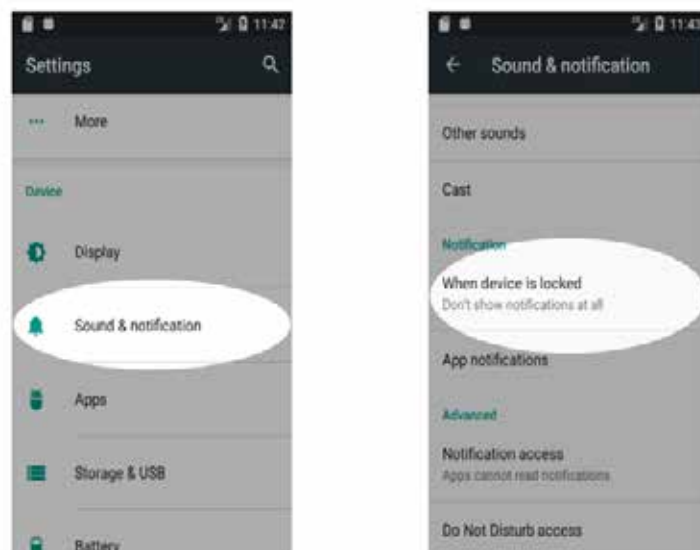
Decide and review which data should be backed up in the cloud

Not synchronizing the applications restricts the data sent to cloud servers like Google Drive. For example, WhatsApp backs up the data to the Google Drive unencrypted even though it is an application which encrypts the data with end-to-end encryption by default. Review such synchronization settings - you can use the options found in *Settings > Accounts section > [app name]*.

Hide personal notifications

Your phone is locked but an incoming notification reveals who you are chatting with? Is the caller information seen although the PIN is active? If so, it is time to review the notification settings; you can prevent the data from being seen when the screen is locked (only viable for newer versions).

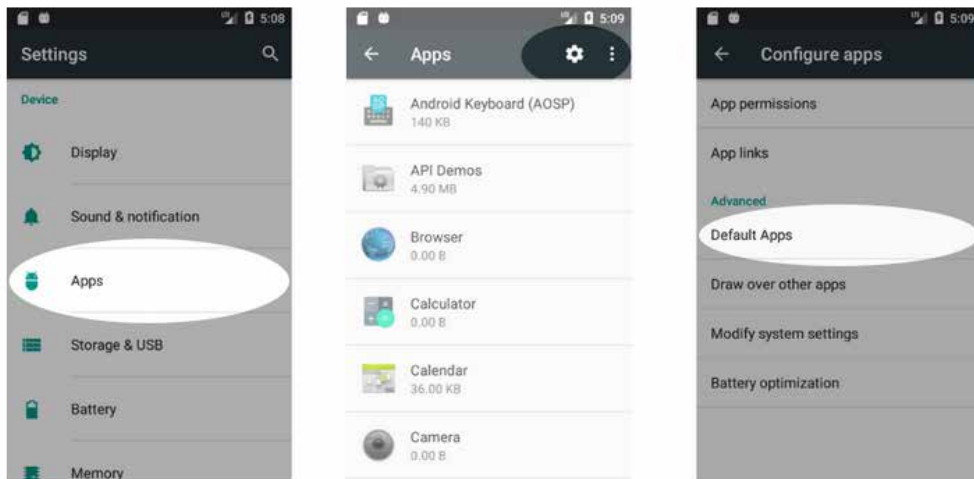
Settings > Sound & notification



Review the default applications

Which application does your device use by default for sending a text message? Or when you click a link, which web browser will display the website by default? Review such default application settings. You can change the default applications with trusted applications.

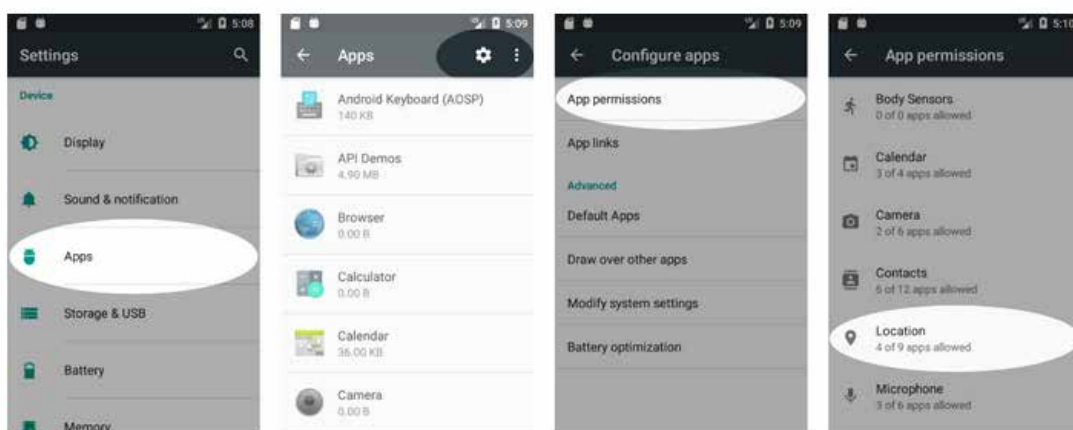
Settings > Apps > [icon] > Default



Do not share your location with the apps!

Control which applications can access your location information.

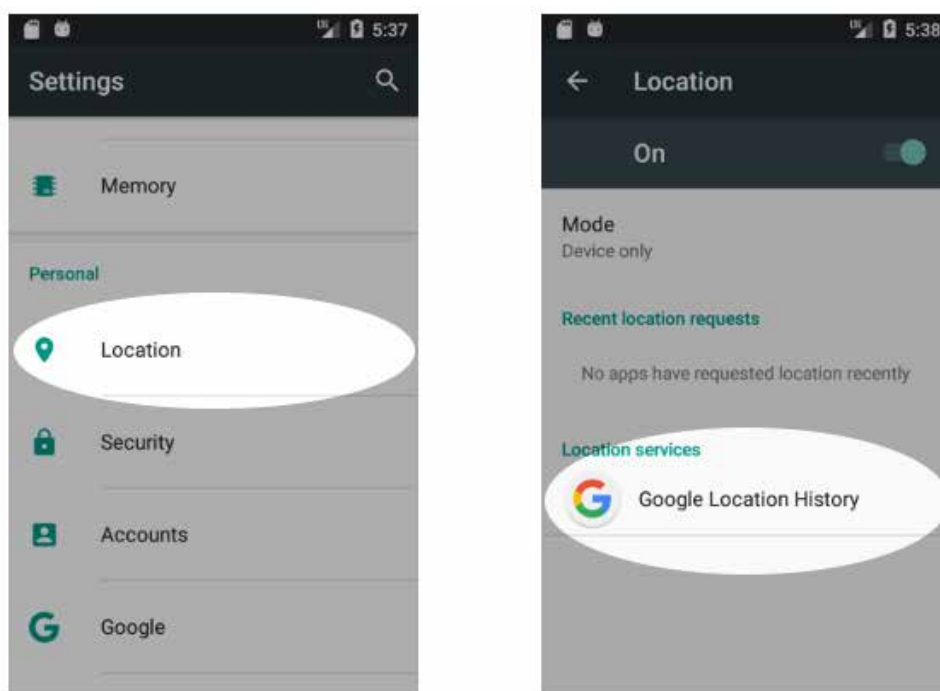
Settings > Apps > [icon] > App permissions > Location



Do not share your location information with Google

In the step above, we have determined whether the applications will access our location information or not. These applications/services include Google. ADINT, a research made by the University of Washington reveals how you can watch someone step by step using mobile advertisements with a budget of approximately a thousand dollars¹. You can prevent this by inhibiting the applications from accessing your location information. With the following menu options you can check whether your applications can access your location:

Settings > Location > Google Location History



You can install a custom Android version on your device

You can install a custom Android version such as CyanogenMod based LineageOS etc. However, technical information is required for this process. Please note that after installation, your device may no longer be covered by the warranty.

You may use DuckDuckgo or Startpage as default browser

You can use DuckDuckgo or Startpage as the default browsers. They are browsers which do not store your search history and care for your privacy.

Reset MAID (Mobile Advertising ID)

MAID is a cookie-like value uniquely assigned to devices, used in ad tracking. In mobile advertising, ads can be targeted using MAID. For instance, ads can be set so that only those with a specific MAID value can see it. The ad display information, date and time can be correlated with the location information and your location can be found. You can be watched step by step.

Reset the MAID value by clicking *Google Settings > Ads > Reset advertising ID*

P.S. Android Privacy Tips blog post published by DuckDuckGo has been utilized.

Source: <https://spreadprivacy.com/android-privacy-tips/>

¹ <https://adint.cs.washington.edu/>

Signal Intelligence

Signal Listening

and

Analysis Methods

Today, when wireless communication takes up every aspect of our lives, the signals emitted by many devices surround us. Radios, mobile phones, wireless modems, Bluetooth devices, communication satellites, GPS satellites, and many other devices are continually emitting signals at different frequencies. It is possible to listen, analyze and even decode these signals with the necessary hardware and software.

With the help of computers, much different hardware can be used to listen for signals. The equipment is available in many different options, from professional equipment to mini USB devices. Using the Realtek RTL2823U chipset, the RTL-SDR hardware is one of the most inexpensive and efficient hardware solutions with

a price of \$ 20. This equipment is commercially available to monitor terrestrial TV broadcasts via computers and can operate at all frequencies between 24 and 1766 Mhz. With this feature, in addition to terrestrial TV broadcasts, FM-AM can listen to radio channels, police, lifeguards, firefighters, coast guard radios, amateur radio frequencies, GSM signals, and many satellite signals.

To benefit from this equipment efficiently, the antenna must be suitable for the data to be received. The RTL-SDR equipment can only listen to a signal, not to transmit signals. More expensive alternative equipment such as HackRF and BladeRF can perform both receive (Rx) and send (Tx) operations.



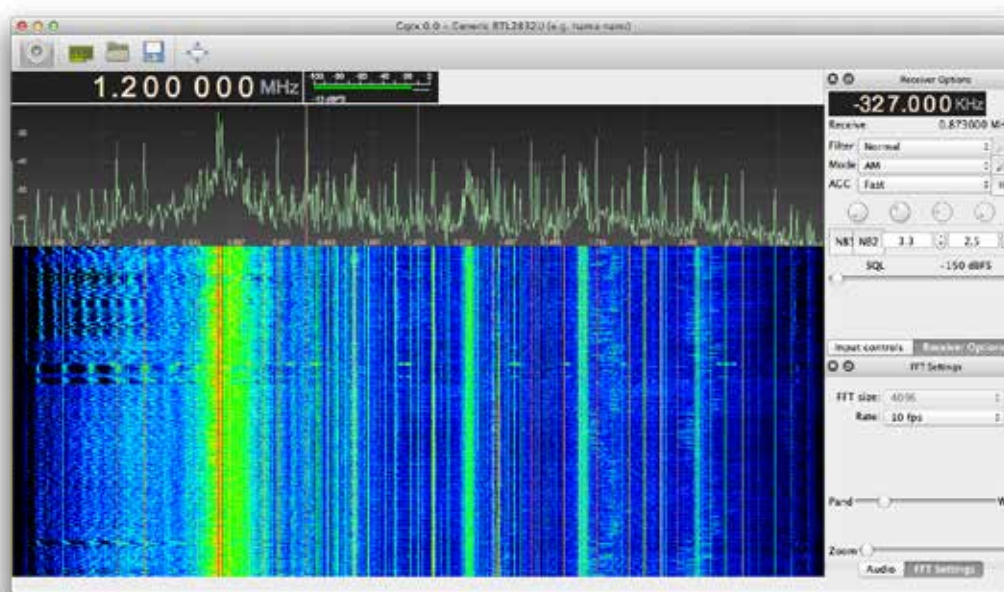
(RTL-SDR Hardware Images)

Some of the things that can be done with RTL-SDR Hardware

- Police, ambulance, fire radio, and EMS communication can be heard (legally public frequencies)
- Air traffic control conversations can be listened to.
- The locations, speeds and direction information (ACARS) of the aircraft in the air can be heard and displayed on the map.
- By listening to sea traffic, the ship name information, direction, and location information can be processed on the map.
- Amateur radio speakers can be listened to.
- Digital audio communication can be heard and decrypted. (DMR digital radios)
- Wireless security camera, baby monitor, Bluetooth devices such as signals can be monitored.
- POCSAG / FLEX Pager systems can be displayed and displayed as text.
- Satellite imagery and weather information can be obtained from meteorology satellites. (NOAA Satellites)
- Analog terrestrial TV broadcasts can be viewed.
- FM and AM radio channels can be played back.
- GSM signals can be rested and analyzed.
- RF signals can be rested and analyzed.

Listening of signals with software

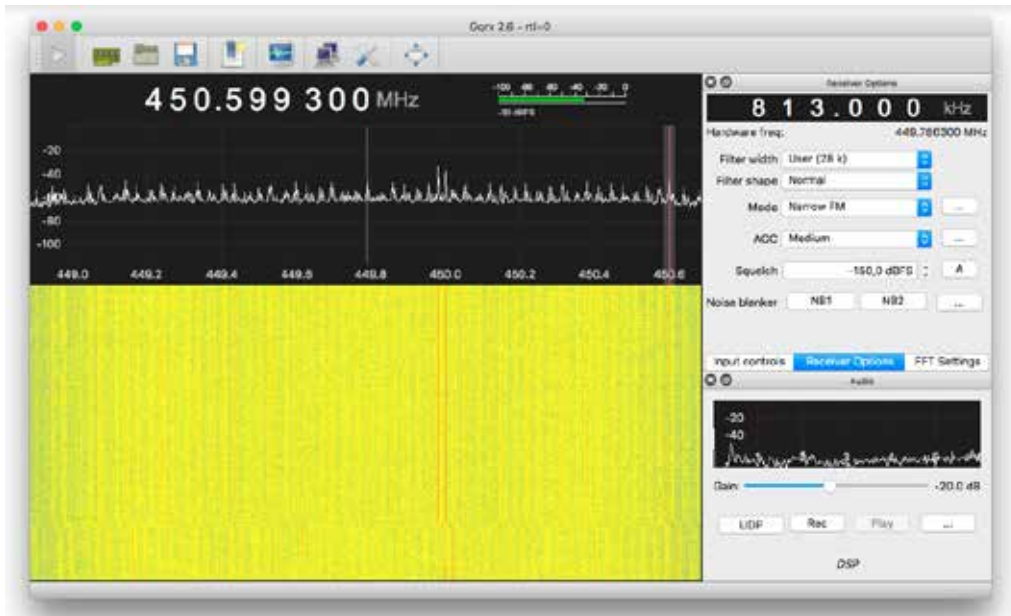
RTL-SDR hardware can run on all operating systems. SDRSharp or Gqrx software can be used to listen to signals simply. The most widely used Gqrx software for Linux and MacOS operating systems. When the desired frequency is reached, listening can be performed if there is an audio communication or Gqrx can be converted to a UDP server and the data of the given frequency can be shared with other applications via the UDP port.



(Gqrx Software Image)

Listening to Police Radios

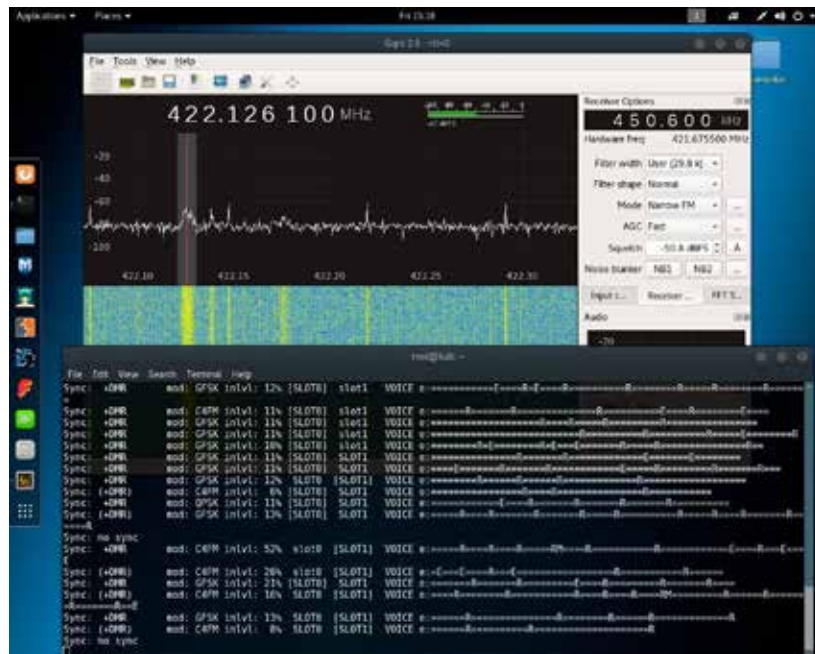
Some radio frequencies used by traffic or public order policemen broadcast publicly, and listening to them is not a legal offense. In general, journalists listen to these frequencies and know about crimes such as traffic accidents or theft. Police radio frequencies can be found on the internet.



(Gqrx Police Radios image)

Listening to Digital Radios

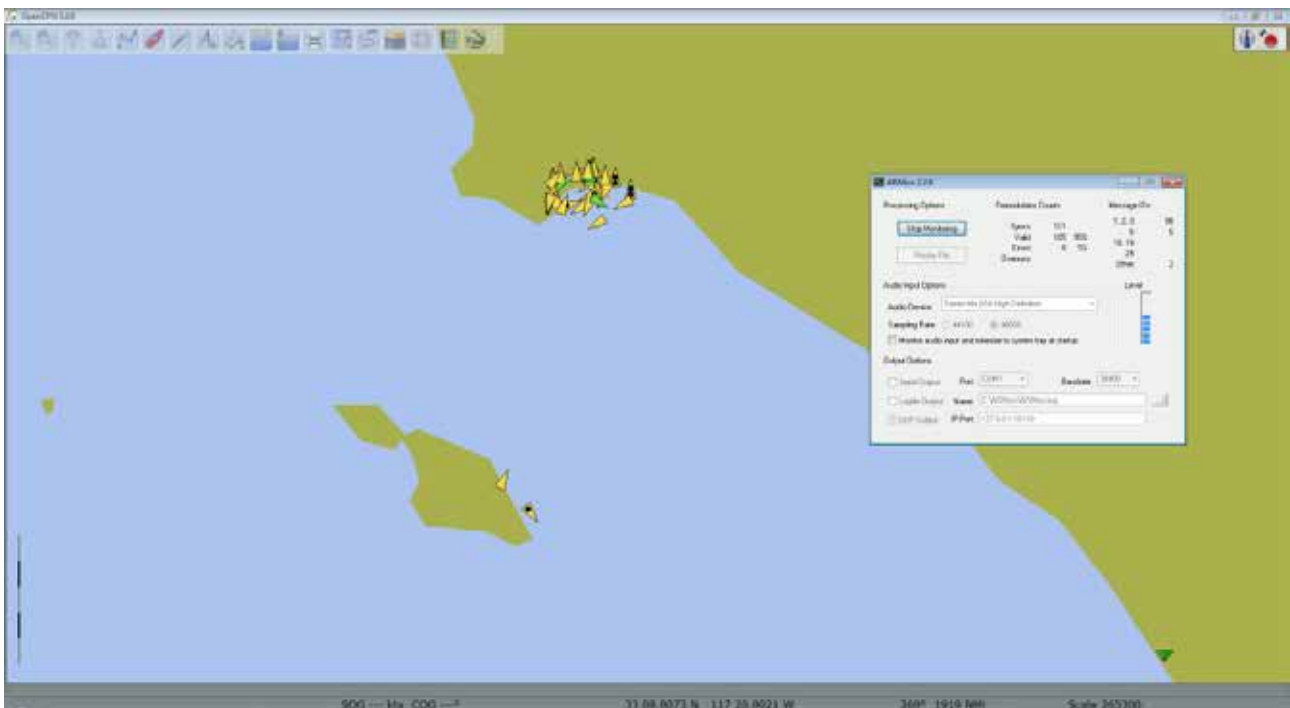
Police, fire brigades, ambulances or even security guards in shopping malls have now started to use digital radios due to security and other issues. However, these digital conversations called DMR can be decoded with some software.



(Decoding digital radios with Gqrx and other software)

Monitoring and Analysis of Ship Traffic

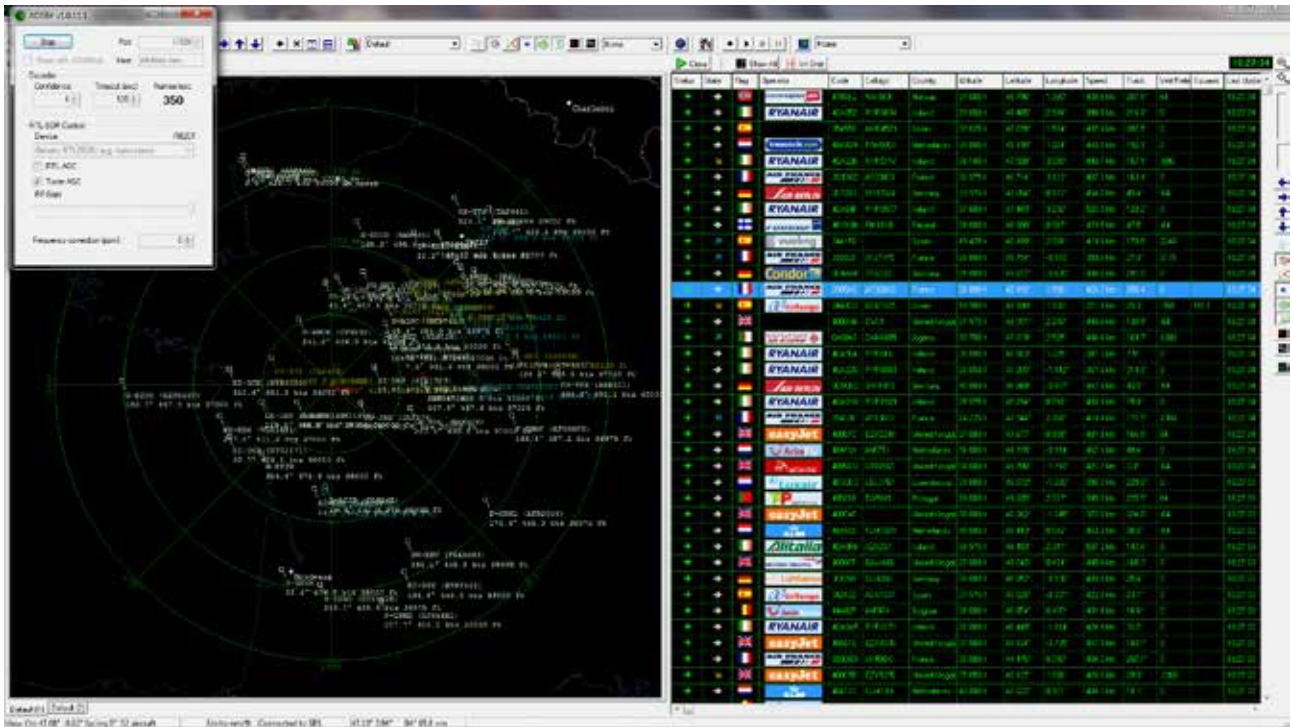
Each ship over a certain size and carrying passengers uses the AIS (Automatic Identification System) tracking system. VHF is constantly broadcasting at 161.975Mhz and 162.025Mhz channels with GMSK modulation. In these signal broadcasts, information such as the name of the ship, call code, coordinates, route, speed, ship size, destination port and time of arrival are sent. It is not a crime to listen to these signals or to analyze them with the help of computers, but it is forbidden to broadcast on these external frequencies. This field is full of weaknesses because the signals can be manipulated easily, and the radar of a high-tonnage ship can be displayed as if there was another ship with false AIS signals. AISMON and OpenCPN software help to mark the ships on the map with the signals received.



(Vessel traffic listening and analysis image)

Monitoring and Analysis of Air Traffic

All aircraft ready for take-off or active in the air must transmit their information to the stations on the ground via signals. This information includes many data such as speed, altitude, location, destination, and direction of travel, as in marine traffic and is called ADS-B. This data is propagated by a device called the transponder. The 1090Mhz frequency can be listened and analyzed and all the airplanes that were on us at that time can be displayed.



(Air Traffic Listening and Analysis image)

Listening and Analyzing Satellite Signals

Hundreds of different satellites pass through each day, and many of them transmit data to the ground via signals through which they pass. NOAA satellites designed to be used in the meteorological field openly broadcast these signals to the whole world. Many NOAA satellites are continually circulating in different locations in orbit and can be instantly displayed on the internet. Almost all satellites can be displayed at <http://www.n2yo.com>.

Turnstile, Quadrifilar Helix, V-Dipole or Double Cross type antenna must be used to receive NOAA satellites efficiently. The antenna that comes with the RTL-SDR equipment is not sufficient.



Turnstile



Quadrifilar Helix (QFH)

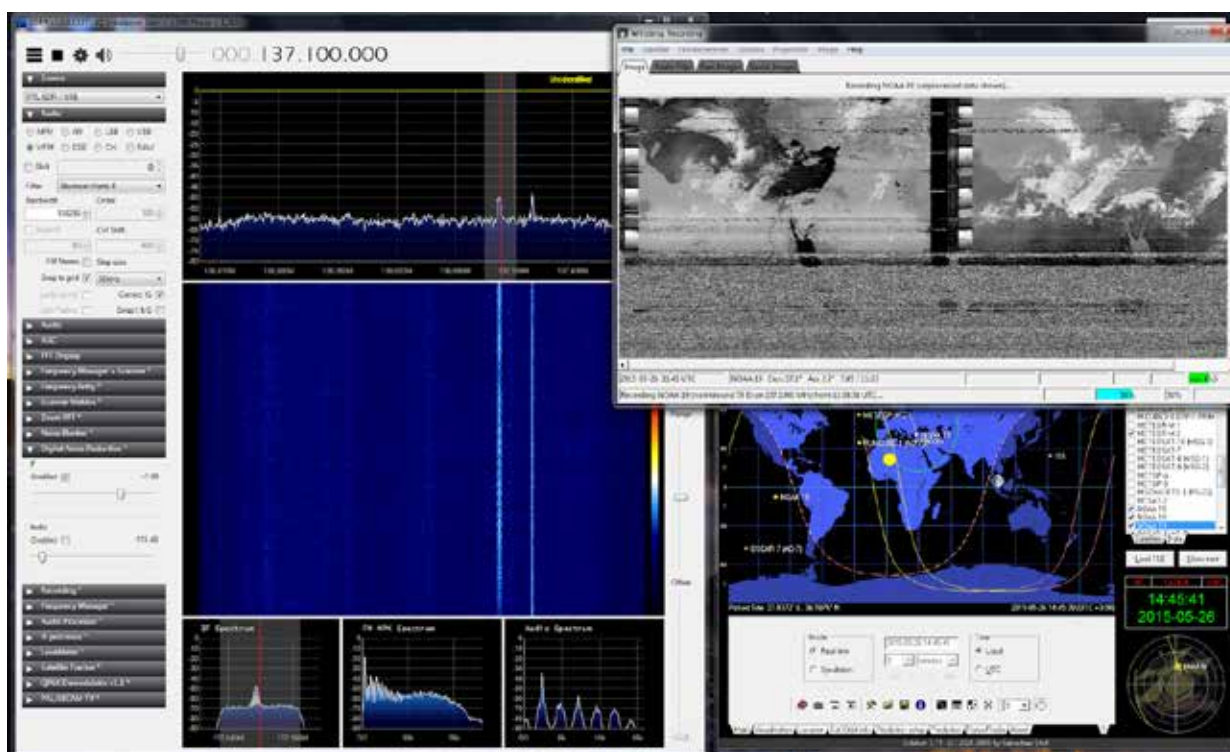


V-Dipole



Double Cross Antenna (DCA)

When the NOAA-19 satellite passes through, it sends signals at a frequency of 137.100Mhz with SDRSharp software, and the signals are transferred to WXtoImg software and converted into photographs.



(Satellite Listening and Analyzing image)

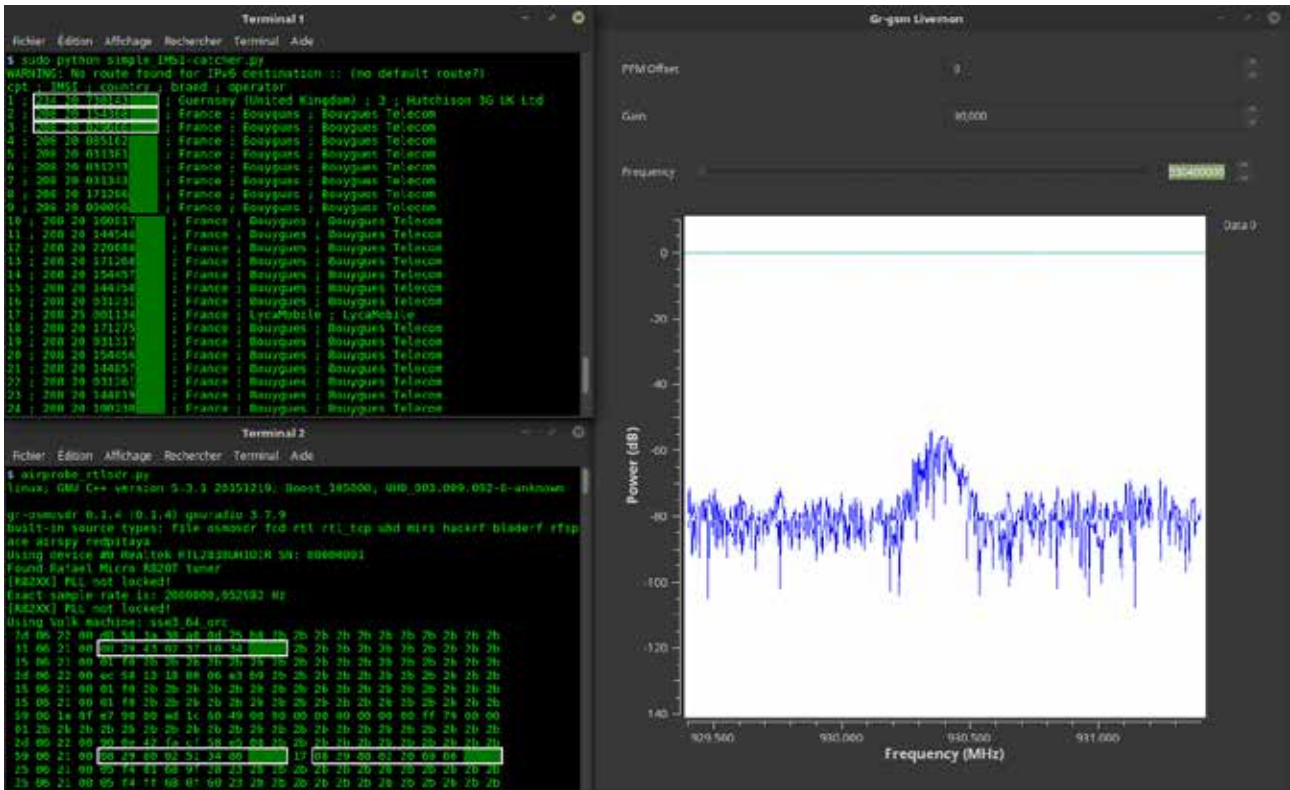
Listening and Analyzing GSM Base Stations

With the RTL-SDR hardware, just like what mobile phones do, GSM signals can be listened to. Each wireless communication system has a standard. The systems used for GSM communication are still ancient and contain many vulnerabilities. By listening to signals easily and analyzing these signals using some software, threats may arise.

Instead of reading conversations or chats in GSM communication, it alone creates a significant vulnerability to listen to the signals and to learn the codes which belong to every mobile phone user called IMSI. Kevin Mitnick, the famous hacker whom dozens of books and films have been dedicated to, managed to escape from the FBI for a long time. Mitnick had acquired the complete IMSI code of the FBI staff by using his social engineering brilliance. He was always listening to the base stations on his computer and checking his IMSI numbers. If the IMSI code of one of the FBI employees entered the base station around him, his computer would give him a warning, and he was able to get away and escape. This method used in the early 1990s can still be used. Mitnick, who listened to GSM signals, also got captured in a similar manner. Tsutomu Shimomura, a computer security expert, set up a counterfeit GSM base station to catch Mitnick, allowing Mitnick to communicate and record all his communications.

When combined with social engineering and GSM weaknesses, open-ended and impossible to prevent threats arise. If the GSM technologies are not overhauled, these vulnerabilities will continue to pose significant threats.

With RTL-SDR and Gr-GSM software, IMSI numbers can be displayed by analyzing the signals from GSM base stations.



(Listening and Analyzing GSM Base Stations - 1)

```
root@kali: ~/Desktop/IMSI
File Edit View Search Terminal Help
root@kali:~/Desktop/IMSI# python IMSI_yakalama.py
Nb IMSI ; TMSI-1 ; TMSI-2 ; IMSI ; Ulke ; Marka ; Operator ; MCC ; MNC ; LAC ; CellId
1 ; ; ; 286 02 0325755075 ; Turkey ; Vodafone ; Vodafone Turkey ; 286 ; 02 ; 53410 ; 9151
2 ; ; ; 286 02 3870235819 ; Turkey ; Vodafone ; Vodafone Turkey ; 286 ; 02 ; 53410 ; 9151
3 ; ; ; 286 02 8240268612 ; Turkey ; Vodafone ; Vodafone Turkey ; 286 ; 02 ; 53410 ; 9151
4 ; ; ; 286 02 2450086628 ; Turkey ; Vodafone ; Vodafone Turkey ; 286 ; 02 ; 53410 ; 9151
5 ; ; ; 286 02 3212217628 ; Turkey ; Vodafone ; Vodafone Turkey ; 286 ; 02 ; 53410 ; 9151
6 ; ; ; 286 02 8570218508 ; Turkey ; Vodafone ; Vodafone Turkey ; 286 ; 02 ; 53410 ; 9151
7 ; ; ; 286 02 0555990028 ; Turkey ; Vodafone ; Vodafone Turkey ; 286 ; 02 ; 53410 ; 9151
8 ; ; ; 286 02 8370281770 ; Turkey ; Vodafone ; Vodafone Turkey ; 286 ; 02 ; 53410 ; 9151
9 ; ; ; 286 02 0337418449 ; Turkey ; Vodafone ; Vodafone Turkey ; 286 ; 02 ; 53410 ; 9151
10 ; ; ; 286 02 4770193189 ; Turkey ; Vodafone ; Vodafone Turkey ; 286 ; 02 ; 53410 ; 9151
11 ; ; ; 286 02 4960228905 ; Turkey ; Vodafone ; Vodafone Turkey ; 286 ; 02 ; 53410 ; 9151
12 ; ; ; 286 02 3650041109 ; Turkey ; Vodafone ; Vodafone Turkey ; 286 ; 02 ; 53410 ; 9151
```

(List of Listening and Analyzing GSM Base Stations - 2)

CALL
FOR
PAPERS

ARKAKAPI

Do you want your article to be published on Arka Kapi Magazine? Submit now to be featured in the next issue! Your article can be of any title as long as it fits to the cyber security context. Make sure it's an original article that isn't previously published elsewhere.

Email your articles to:
editor@arkakapimag.com

FEEDBACK

Got any feedback about Arka Kapi Magazine? Found a bug? Want us to add or remove something? Let us know!

follow us

Don't miss the news!



arkakapimag