The History of Information Security

Martin Pozděna
Security in Telecommunications
Technische Universität Berlin
March 31, 2015

Email: martin.pozdena@campus.tu-berlin.de

***Abstract*—Objective of this paper is to briefly cover the history of information security including the most influential research as well as examples of successful and failed real-world implementations. Logical structure of the paper consists of several sections each covering single domain of information security. Initially paper provides fundamental introduction into the theory of information security. Subsequent sections cover the following subfields of information security (in respective order): Cryptography, Hardware Security, Network Security, System Security and Application Security. Paper finally concludes with the most recent trends in the field of information security and possible future outlook.**

I. Introduction to the Information Security

Invention and rapid development of general purpose computing starting in the second half of the 20th century brought about a great technological, economical and social shift. The creation and the use of information is getting an ever increasing importance in our everyday life since then. Given the importance information gained over the time it is vital for any computer system that is used to process it to do it in the secure and anticipated manner.

The fundamental component of information security is so called CIA triad — confidentiality, integrity and availability. Confidentiality requires the data to be only accessible to those who are authorized to read them, thus preventing the unauthorized disclosure. One possible way of ensuring confidentiality is to encrypt the data before storing it or before transmitting it over an insecure channel. Consequently, only the authorized person who posseses the correct decryption key is able to read those data. This simple approach, however, brings about a new problem of ensuring that decryption key is not disclosed to any unauthorized party. [1]

The integrity component means that no unauthorized person is able to modify the information. Integrity consists of several distinguishable attributes. Data integrity ensures that the data itself has not been modified in an unauthorized way, while the origin integrity distinguishes whether the originator of that data is really the party it claims to be (usually called authentication). There are different mechanisms how to ensure integrity. In case that breach of data integrity is absolutely unacceptable in the target computer system, prevention mechanisms need to be employed. Example of such a system is online banking. In this case it is absolutely necessary that integrity of each account balance information is ensured, thus preventing any unauthorized party in changing it in unauthorized manner. On the other hand, there are situations in which prevention mechanisms are unfeasable to use. In case of two systems communicating over the network it would be impractical to employ prevention mechanisms, but it is enough to detect that breach of data integrity occured so the system can request data retransmission. [1]

Availability component represents the necessity of data being available to the legitimate users. For instance well-known denial of service attack attempts to discrupt the service by submitting more requests for a given resource than the system can handle. Consequently, the resource appears unavailable even for a legitimate users, thus the availability of the information is not provided. [1]

Matt Bishop defined three underlying computer security concepts in his paper "What is Computer Security?" as follows: [2]

- Security Requirements
- Security Policies
- Security Mechanisms

Paper explains that each system being it in possesion of a company, university or government has distinct requirements for the computer security. Those are simply called security requirements and are represented by common language. In case of a company it might define that confidentiality of trade secrets needs to be ensured under any circumstances or that the information portal for its customers needs to be available over the Internet 24 hours a day. [2]

Security policies are based on the security requirements and states which actions and system states are allowed and which are not. Based on the example above mounting unencrypted personal flash drives on the system which hosts the trade secrets would be prohibited by company's security policy. System which only resides in the allowed system states and only performs allowed operations is secure. If any of the previously mentioned condition is not met the system is nonsecure. [2]

Finally, the security mechanisms are means how to enforce the security policies. Mechanism can be either technical or procedural. Following the example from previous paragraphs, technical security mechanism would prevent users from mounting flash drive that is not encrypted on the system. On the other hand procedural mechanism might require all employees of a company not to bring personal computing equiment including flash drives into the workplace. Security mechanisms might under certain circumstances fail to enforce the security policies correctly. It can be either due to the incorrect configuration

of the security mechanism or due to the presence of some programing error (for example buffer overflow). [2]

While the introduction of the paper covers the basic concepts of computer security which does not change much over the time, the following sections aim to discuss the history and consequences of some important milestones in the field of computer security. Sections are divided according to the distinct subfields of IT security as: Cryptography, HW Security, Network Security, System Security and Application Security.

II. Cryptography

A. Pre-computer era

People's desire to change transmitted messages in the way that only intended recipient and nobody else is able to read it is considerably old. The oldest cryptographic systems are generally referred to as classical cryptography and were developed well before the invention of a computer, generally requiring only paper and pen for both encryption and decryption. Most famous example of classical cryptosystem is substitution cipher, whose fundamental idea is that each letter of plaintext is substituted with predefined letter during the encryption process. Decryption is then simply done by the opposite process of reverse substitution of letters in received ciphertext. Caesar cipher is one of well-known substitution ciphers. Another example of classical cryptosystem is transposition cipher which does not change the letters of the message but its positions. Simple implementation is to write plaintext message into the matrix, transpose the matrix and consequently read the ciphertext from this transposed matrix. Classical cryptosystems are almost never used nowadays as they proved to be very easy to break over the time.

Increased interest in cryptographic techniques before and during both world wars brought about the invention of more sophisticated electromechanical encryption machines. The course of a history was in many cases altered by the ability of countries to keep their sensitive information confidential. Arguably the most prominent example of the importance of cryptosystems during the world wars is the German encryption machine Enigma. Enigma machine was used during the Second World War by Axis armies to encrypt their confidential radio communication. It was successfully broken by cryptanalysts in Bletchley Park, United Kingdom and allowed Allied forces to read sensitive military messages of its adversary. According to the Supreme Allied Commander Dwight D. Eisenhower this fact was decisive to the Allied victory in the Second World War and it shortened the war by few years. [3]

B. Modern cryptography

Rapid development of Information and Communication Technologies following the Second World War set up the environment for the inception of modern cryptosystems which uses general purpose computers for encryption and decryption process. Firstly, the invention of packet-switching networking, where message is fragmented into the small chunks (packets) to be transmitted over the network and then reassembled at the destination set new requirements for ensuring confidentiality and integrity of the messages. Packet-switching network allows every packet to take different path from source to its destination, so every party on that path is able to eavesdrop the message or alter it. Secondly, the invention of email communication and its aspiration to gradually replace regular mail as more efficient way of communication needed to solve major challenges. It needed to ensure the confidentiality and integrity of messages the same way as the secrecy of correspondence is guaranteed in the regular mail. Moreover, it needed to provide a way to electronically sign the messages in order to assure the authenticity and non-repudiation. [4]

First data encryption algorithm to be approved by United States as Federal Information Processing Standard in 1977 and widely deployed was DES (Data Encryption Standard). [5] It was symmetric-key algorithm (using the same key for encryption and decryption) based on Feistel network. It uses 56 bit long key, which is too short to hold against the bruteforce attack nowadays. It was first superseded by TripleDES in 1999 and withdrawn at all in 2005. [6] The symmetric-key algorithm approved as Federal Information Processing Standard and widely deployed nowadays is Rijndael also called AES (Advanced Encryption Standard). [7] AES algorithm is widely used nowadays and has not been publicly broken yet.

Suppose that we have reliable symmetric-key cryptographic algorithm that can ensure the confidentiality of the data providing the key is kept secret. If we want to send an encrypted message (email) to some person we first need to exchange the encryption key using some secure channel. Therefore, symmetric-key algorithm usefulness is minimized if such a secure channel is not available. Moreover, neither DES nor AES standard tries to address the issue of electronic signatures. This problems were addressed by W. Diffie and M. E. Hellman in their influential paper "New Directions in Cryptography" which was published in 1976. They propose the usage of two new cryptographic systems: [4]

- public key cryptosystems
- public key distribution systems

Public key cryptosystem was defined as a pair of public key E_k and private key D_k. E_k is inverse of D_k, but given the knowledge of E_k it is computationally infeasible to derive D_k from it. This property allows E_k to be made public while keeping private key D_k secret. Consequently, when somebody wants to send a message it can be encrypted using public key E_k and sent over the insecure channel. Only the party in possession of D_k is able to decrypt such a message afterwards. Finding such public key cryptosystem would solve the issue of secret key distribution. Each communicating party would generate its own key pair E_k and D_k, making E_k public and keeping D_k secret. When somebody wants to send a message it uses the public E_k of intended receiver for encryption. This way the need for using secure channel is eliminated as D_k is never communicated anywhere and E_k should not reveal any useful information for ciphertext decryption. [4]

Finding such cryptosystem would also solve the problem of electronic signatures. If communicating party would like to sign the message it can encrypt it using its own private key D_k. When such a message is received by its addressee it can try to decrypt it with public key E_k belonging to the sender. If encryption is successful it means that this message was encrypted using private key D_k belonging to that public

key E_k. As there should be only one person in possession of that private key D_k and we know who it is the person is authenticated. [4]

Diffie and Hellman illustrated possible public key cryptosystems as generating two inverted matrices where one is used as public key and second as private key. Encryption and decryption can be done as multiplication of vector (message) with those matrices. This way the encryption and decryption would require n^2 operations where n is length of vector (message) and matrix inversion would require approximately n^3 (the more efficient Coppersmith-Winograd algorithm for matrix inversion was not known in 1976 [8]). This would theoretically mean that if you use matrices that are large enough it would be computationally feasible to perform encryption and decryption but not the matrix inversion. Nevertheless, the authors admitted at the time that this example is of no practical use and usable public key cryptosystem was yet to be discovered. [4]

Another concept presented in the paper is public key distribution systems. This concept allows secure exchange of key over the insecure channel. Diffie and Hellman proposed key exchange algorithm in their paper that is used until today. It is based on the difficulty of computing logarithm modulo q, where q is a prime number. It is simple to compute Y in the following equations, but difficult to do reverse operation computing X ouf of Y: [4]

$$Y = \alpha^X \bmod q$$
$$X = \log_\alpha Y \bmod q$$

When two parties want to establish common key K over insecure channel they first exchange α and prime number q. Subsequently, they generate X_i and X_j that are less than q and then exchange $Y_i = \alpha^{X_i} \bmod q$ and $Y_j = \alpha^{X_j} \bmod q$ over insecure channel. Common key K is than computed as follows: [4]

$$K = Y_i^{X_j} \bmod q$$
$$K = Y_j^{X_i} \bmod q$$

Even if attacker manages to eavesdrop both Y_i and Y_j it would be computationally infeasible for him/her to derive K out of it.

It did not take a long time for usable public key cryptosystem to be discovered. Rivest, Shamir and Adleman presented it in paper called "A method for obtaining digital signatures and public-key cryptosystems" roughly two years later in 1978. Presented public key cryptosystem became widely used and well-known under the name of RSA. Its security is based on the complexity of factoring composite number n. Cryptosystem can encrypt messages represented as a number from 0 until $n-1$. Encryption of message M and decryption is done as follows: [9]

$$C = M^e \bmod n$$
$$M = C^d \bmod n$$

Therefore (e, n) is public key in RSA cryptosystem and (d, n) is private key. n is computed as product of two large random prime numbers p and q. As factoring n to prime numbers p and q is too computationally complex p and q are effectively hidden even when n is public. Public exponent e that needs to be coprime with $(p-1)\cdot(q-1)$ is also picked randomly. Private exponent is multiplicative inverse of e modulo $(p-1)\cdot(q-1)$: [9]

$$e \cdot d \equiv 1 \pmod{(p-1)\cdot(q-1)}$$

Although RSA was first published public-key cryptosystem and is possibly the most well-known nowadays there have been more public-key cryptosystems described over the time like ElGamal or Elliptic curves based cryptography.

Authors of RSA public key cryptosystems encouraged cryptanalysts to examine the security of RSA algorithm and to try to break the cryptosystem. [9] Nevertheless, RSA has not been broken until today and it is therefore considered to be secure if reasonably large key length is used. Although nowadays widely deployed cryptographic algorithms like RSA and AES are considered strong enough there have been several prominent attacks and failures of cryptosystems in the past. In some cases poor implementation of cryptographic algorithms allowed revealing the key through successful side-channel attacks which are covered in detail in the following section III. History has also seen the attempts to create own cryptosystems and trying to keep the algorithm as well as its implementation hidden in the devices. GSM cryptosystems like A5/1 and A5/2 were both reverse engineered from the devices and later broken, more information is provided in section IV of this paper.

III. Hardware Security

A. Introduction to HW Security

Hardware security of computer devices became a concern in cases where such a device can get into the possession of potential attacker. Typical use-case where hardware security implications need to be taken into account are smart cards. Plastic cards with a chip are for example used as contactless payment cards, authorization cards for pay television, phones SIMs or as a part of biometric passports. Authorization cards for pay television need to be very hard to clone, otherwise it is possible to copy the card and sell the counterfeits illegally so multiple people can watch the program in different locations with just one subscription paid. Oliver Kömmerlich and Markus Kuhn argue in their paper "Design principles for tamper-resistant smartcard processors" that almost every pay television conditional access system based on smart cards was successfully reverse engineered and copied between 1994 and 1999. [10] Another example of computer systems that needs to be extremely hard to tamper with or reverse engineer are computer chips and electronics in military systems.

Multitude of different hardware security related attacks were described in the past. They can be generally divided into the following classes:

- Non-invasive
- Semi-invasive
- Fully-invasive

Non-invasive class represent the attacks that do not tamper with physical integrity of examined device. It contains all possible types of side-channel attacks like power analysis or timing attacks, mostly pioneered by Paul Kocher in late nineties. [11] [12] Another non-invasive attack is glitching which is made possible due to the fact that clock and supply voltage for a smartcard is usually provided by the environment. Voltage glitch for example reduces the amount of supply voltage at the specific point which might affect the value that is read from memory. Clock glitch on the other hand changes the clock provided to the smartcard in order to change the execution sequence of a program running on the chip. Other way how to non-invasively alter the operation of a chip is to expose it to the non-standard operation environment like extremely low or high temperatures or x-ray radiation.

Semi-invasive class represent the class of attacks where computer device (chip) is taken out from the package (plastic card), but its internal structure and functioning is not permanently altered. Internal structure of such a chip can be consequently examined and reverse engineered for example using microscope or photonic emission analysis. Moreover, faults changing the behavior of a chip can be introduced by microprobing or laser stimulation.

Fully-invasive class contains attack vectors that physically extract the chip from the casing and then tamper with the internals of the chip. This could be used in order to circumvent security checks, for example by connecting the wire that outputs the success or failure of security check by true (1 – high voltage) or false (0 – ground) to supply voltage, therefore being always true. It can be also utilized for changing read-only memory of the chip and more.

B. Side-channel attacks

The first influential side-channel attack described by Paul Kocher was published in 1996 in paper "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems". The fundamental idea exploits the fact that execution time of cryptosystem implementation differs according to the key and the input data which can lead to the exposure of secret key. [12] Let's assume simple smartcard that is used as an authentication token of its owner (for instance to authenticate the person before entering the building). Such a smartcard needs to fullfil following two criterions:

- identify itself uniquely to the legitimate reader
- being hard to copy

Therefore smartcard that simply sends its ID to the reader fails to fulfil the second criterion as attacker can read the ID and easily create counterfeit card sending the same ID. This problem can be elegantly solved using public key cryptography explained in section II. Pair of RSA public and private key is generated for each card. Public key is stored in the reader with information to which card it belongs. Private key is stored in the memory of the card. During authentication process reader first sends some random nonce to the card, card receives it, signs it with its private key and sends the signed nonce back to the reader. Reader can now verify that nonce was signed by the correct private key. This is reliable way of authentication that does not require private key to be exposed during the authentication process.

As it was already explained in section II, electronic signature using RSA is encryption of the message with the private key. This process needs to calculate $S = M^d \bmod n$. Modulus needs to be reasonably large number in order to ensure that it is not computationally feasible to factor it to its prime numbers. Modular exponentiation with large modulus is usually implemented as some form of square-and-multiply algorithm. Calculation of $y^x \bmod n$ where x is ω bit long exponent is done as follows: [12]

1: $s_0 \leftarrow 1$
2: **for** $k \leftarrow 0$ upto $\omega - 1$ **do**
3: **if** (bit k of x) is 1 **then**
4: $R_k \leftarrow (s_k \cdot y) \bmod n$
5: **else**
6: $R_k \leftarrow s_k$
7: **end if**
8: $s_{k+1} \leftarrow R_k^2 \bmod n$
9: **end for**
10: **return** $R_{\omega-1}$

If exponent bit is 0 only square operation is taken, while when exponent bit is 1 both square and multiply operations are carried out. Paul Kocher identified that if we have enough timing measurements for multiple messages being signed we can extract the secret key (exponent). [12] Gaining multiple timings for different messages is not a problem as we can send different nonces to the same card and measure how long it takes to sign the nonce. Given that we have multiple samples we can easily calculate mean value and variance.

Each timing measurement can be expressed as $T = e + \sum_{i=0}^{\omega-1} t_i$, where: [12]

- t_i - multiplication and squaring time in each iteration
- e - time of everything else (loop overhead etc)
- ω - bit-length of x (private key)

The fundamental idea of the attack is that we can guess first b bits of the exponent x, emulate square and multiply algorithm for those b bits and measure how long it takes for each nonce to complete. Meaning that $T_{x_b} = \sum_{i=0}^{b-1} t_i$ can be emulated. If we subtract emulated times from the measured timings we get $T - T_{x_b} = e + \sum_{i=b}^{\omega-1} t_i$. Doing this over all measurements for correct guess b would decrease the overall variance from $Var(e) + (\omega) \cdot Var(t)$ to $Var(e) + (\omega - b) \cdot Var(t)$. On the other hand guessing b bits, but only having first c bits guessed correctly would increase the overall variance to $Var(e) + (\omega + b - 2c) \cdot Var(t)$. Given the fact that our sample is large enough to correctly distinguish between correct and incorrect guess based on the change of overall variance we can expose the whole secret exponent (private key) bit by bit. [12]

Some other interesting timing attacks were introduced since Paul Kocher's initial revelation. Song, Wagner and Tian presented in 2001 the theoretical approach that can reveal some information about data transferred over SSH. [13] SSH in interactive mode sends every keystroke in separate packet. The fundamental idea is that people tend to type different key combinations in the different pace which can reveal some interesting information only by measuring the delays between

individual packets on the network. Another interesting timing attack was published by Brumley and Boneh in 2005 in paper "Remote Timing Attacks are Practical". [14] Until then it was generally expected that timing attacks can only be utilised locally. Authors showed that it is possible to employ successful timing attacks towards OpenSSL over the network.

Apart from timing attacks, other side-channel attacks were also proved to be useable. Paul Kocher, Joshu Jaffe and Benjamin Jun presented in their paper "Differential power analysis" in 1999 method that can compromise the private key processed on the chip based on the differences of chip power consumption. [11]

C. Physically unclonable functions

Pursuit of creating chips that can uniquely identify itself to the reader while not being copiable lead the research community to evaluate the possibility of utilizing the physical properties of silicon integrated circuits. The concept named physically unclonable functions (PUFs) was introduced in the paper "Silicon Physical Random Functions" in 2002. It describes that manufacturing process of each integrated circuit introduces enough deviation of individual physical characteristics of each circuit. Therefore, it should be possible to uniquely authenticate that circuit based on its physical properties. It describes PUFs as a function embodied in the physical device that maps challenges to the responses and has the following properties: [15]

1) Easy to evaluate — For every challenge the response is computed in timely manner.
2) Hard to characterize — It is unfeasible to describe how PUF maps challenges to the responses.
3) Manufacturer resistant — It is unfeasible to manufacture two PUFs that maps responses to the challenges the same way
4) Controlled — PUF can only be accessed through interface that was designed for it, meaning that any attempts to access the PUF in other way (for example by fully-invasive techniques which would inevitably change the physical properties of a chip) leads to the destruction of PUF

When each device with PUF is manufactured certain sample of challenges is sent to it and responses are recorded. Consequently, when it is desirable to authenticate the device some subset of those challenges is sent to the device observing how it responds. If responses matches to those captured during the initialization phase the device is authenticated.

Several possibilities how to implement PUF were described over the time. Arbiter PUF is one of them, it consists of several boxes connected linearly to each other. Signal is sent through the boxes as it is shown in the figure 1. Each box takes one bit of a challenge as its input. If the corresponding bit is 0 the signal takes straight path if it is 1 then the signal takes switch path. Response is then 0 or 1 based on which signal arrives first into the arbiter. [16]

The responses of arbiter PUF and some other PUF constructions were proved to be machine learnable in 2010 [17]

Other possible construction of PUF is memory based SRAM PUF. It utilize the property that each SRAM cell stores

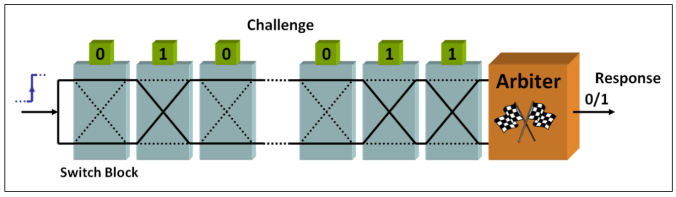

Fig. 1: Arbiter PUF [16]

one bit and its initial content after supplying the voltage is dependent on small differences during the manufacturing. [16] Challenge is than describing which SRAM cells are to be read after the device is turned on and response is the content of those cells. This type of PUF was also proved to be insecure as researchers managed to read out all cells without changing the the PUF properties and subsequently created a copy of it in 2013. [18]

This should not leave an impression that all possible PUF solutions are broken and proved unusable. The research in this field and other PUF solutions is still ongoing and some companies are already producing PUF based hardware security solutions. [19]

IV. Network Security

Computer networking started to develop in early sixties thanks to the research done in packet-switching networks. ARPANET, built on behalf of U. S. Department of Defense, was the first packet-switched network using well-known TCP/IP protocol family. Consequently, U. S. universities and other departments of U. S. government either started to connect to the ARPANET or developed their own networks which finally lead to the network of interconnected networks, Internet, as we know it today. One of the early services that helped the inception of Internet was electronic mail defined in 1972. [20] Internet and computer networking in general started to evolve rapidly, very often leaving security of the information transmitted over it insecure. Some of the early influential application layer protocols developed in seventies and eighties like Telnet, Simple Mail Transfer Protocol (SMTP) or File Transfer Protocol (FTP) were not taking security into account at all. [21] Mentioned protocols offer virtually no protection of confidentiality and integrity of information transmitted. For example telnet which can be used for remote terminal connection sends all data in cleartext. Thus, if telnet is used for remote server administration all sensitive data including passwords and executed commands are being sent over the network without any protection. Obviously such information is easy to intercept by any party having access to one of the links through which message is transmitted.

As already described in the section II, cryptographic mechanisms that could be used for ensuring confidentiality and integrity of transmitted data were known at least from late seventies. Therefore, securing information flowing through computer networks was only matter of applying those cryptosystems in communication protocols. Firstly, it was necessary to find a solution for over-the-network authentication to remote systems. Initially, this issue was approached by sending the cleartext password over the network, which is clearly not an optimal solution for networks where messages

flowing through it can be intercepted. Viable solution of secure network authentication started to being developed in 1983 in Massachusetts Institute of Technology as part of project Athena. Aim of the project was to build campus-wide computer network interconnecting large amount of workstations with dozens servers. Strong network authentication protocol which was developed as a part of project Athena and was made public is known as Kerberos. [22]

Fundamental operation of Kerberos was covered in the paper "Kerberos: An authentication service for computer networks" published in 1994 by two of its co-designers. They describe Kerberos as a system consisting of principals (user), verifiers (application server) and authentication server (AS). The main aim is to securely authenticate principals to verifiers over the network. Each principal and verifier has his own secret password that is shared with AS. Whenever principal wishes to authenticate towards verifier it first sends authentication request containing principal's claimed identity, name of verifier and requested expiration date to AS. AS generates random session key and sends it back to the principal along with assigned expiration time, name of verifier and Kerberos ticket all encrypted using principal's password. Kerberos ticket is structure encrypted with verifier's password containing session key, name of principal and expiration time. As principal's password is known only by himself/herself, he/she is the only one who can decrypt authentication response from AS extracting among others Kerberos ticket and session key. It is worth mentioning that principal is unable to decrypt or change Kerberos ticket as he does not know verifier's password. Principal can now authenticate towards verifier by forwarding him Kerberos ticket along with authenticator. Authenticator consist of present time and checksum and is encrypted using corresponding session key. When verifier receives Kerberos ticket it decrypts it by its password and extract session key and principal name. Subsequently, it decrypts authenticator using that session key and checks the timestamp included in order to prevent replay attacks. If there are no discrepancies along the process principal is securely authenticated towards verifier without revealing any information that would allow eavesdropper on the channel to claim fake identity. [23]

This approach requires user to provide his/her password every time he/she receives authentication response in order to decrypt it. However, Kerberos should provide user-friendly single sign on (SSO) solution where user is asked to provide password only once. SSO functionality is introduced by means of ticket granting server (TGS), which might be hosted by the same machine as AS. During principal's initial login into the system, so called ticket granting ticket (TGT) is issued by AS and it works as Kerberos ticket for TGS. This way principal does not need to communicate with AS every time it needs to authenticate towards new verifier. It authenticates towards TGS using TGT and TGS subsequently issues Kerberos ticket for corresponding verifier. [23] Nowadays, Kerberos system is widely deployed in networks where secure authentication is essential. For instance, Kerberos is used in all Windows domain deployments known as Active Directory.

Security issues of existing application-level protocols started to be addressed as computer networking was getting more and more widespread in the beginning of nineties. Secure Shell (SSH) protocol was introduced in 1995 by researcher from Helsinki University of Technology, Tatu Ylönen. Trigger for his work was security incident that happened within the university when user's passwords were exposed by sniffing attack. [24] SSH gradually replaced Telnet in all places where security is of even the smallest concern. Moreover, secure socket layer (SSL) protocol was developed by Netscape in 1994 in order to help secure vulnerable protocols like HTTP, FTP, SMTP and others. [25] SSL provides point-to-point encryption and let aforementioned protocols to run unchanged on the top of it.

Security protocols ensuring confidentiality and integrity of data being transferred over insecure channel have been found vulnerable due to inappropriate architecture multiple times in the past. For example all SSL protocols of version 3.0 (released in 1996) and lower are considered to be broken by now and its use is discouraged. SSL in version 3.0 can be compromised by POODLE attack which was published in September 2014 by Google security researchers Bodo Möller, Thai Duong and Krzysztof Kotowicz. Architecture of SSL 3.0 contains flaw which allows non-deterministic padding that is not covered by Message Authentication Code (MAC). This allows potential attacker to decrypt one byte of message by sending on average 256 blocks to the server. On the top of that, man-in-the-middle attacker can force communication between client and server to get into fallback mode using old SSL 3.0 instead of newer versions. [26]

Another influential example of failed network security solution is that of Wired Equivalent Privacy (WEP). Boom of mobile devices required individuals and businesses to adopt technology that would allow wireless connection to Local Area Networks (LAN). Wireless LAN technology was thus defined in 1999 and widely adopted. [27] WLAN has one fundamental difference from wired technologies as it is hard to avoid wireless signal unlike cables to stay within the premises of the office or living room. It was therefore essential to accompany WLAN technology with encryption that would ensure security of data transferred over the air. Initial standard defines WEP as recommended encryption technology. WEP is based on RC4 stream cipher and contains several architectural flaws that allowed cryptanalysts to break it over the time. [28]

Nowadays, WEP should be considered completely insecure and was superseded by Wi-Fi Protected Access 2 (WPA2) encryption standard. It is possible to compromise WEP key using low-end laptop with aircrack-ng program within few minutes. [29] It is believed, though no proof for this claim has been made public, that compromise of WEP-secured WLAN was crucial step of one of the biggest personal data breaches in the history. Attackers managed to get access to the network perimeter of TJX Companies Inc. in July 2005 supposedly thanks to compromise of WEP secured WLAN in company stores. Attacker's activities went undetected in the network until the end of 2006. It is believed that between 50–94 millions payment card details and personal data of 451,000 individuals were compromised over the mentioned time period. [30] [31]

Nevertheless, outdated and vulnerable network security solutions does not affect only traditional computer networks. GSM, Global System for Mobile Communications, extremely widespread technology standard for mobile phone cellular networks is known for its weak encryption. GSM was first

used in 1991 in Europe and relies on A5 stream ciphers for ensuring confidentiality of mobile phone calls since then. [32] A5 comes in two versions A5/1 cipher is "stronger" and is used mainly in Europe and A5/2 is intentionally weakened version that was used outside of Europe. [33] It is worth mentioning that description of those two ciphers was never publicly released to allow broad evaluation of cipher strength by independent security experts. It was only in 1999 that A5 cryptosystem algorithms were reverse engineered from one of the devices and publicly disclosed. [34]

Cryptanalysts started to challenge security of published ciphers soon after the disclosure. I. Goldberg, D. Wagner and L. Green managed to break A5/2 within 5 hours, describing attack that needs only $O(2^{16})$ steps. [35] A5/1 proved to be slightly more secure, but it contains some flaws that contributed to its final cracking. A5/1 is cryptosystem with key length of 64 bites, however, it turned out after reverse engineering it that 10 least significant bits of the key are deliberately set to 0. Several possible attacks against A5/1 were published over the time including A. Biryukov's, A. Shamir's and D. Wagner's paper "Real Time Cryptanalysis of A5/1 on a PC". Described attack require one-time preprocessing stage of complexity 2^{48} and subsequently allows to compromise confidentiality of 2 minutes long captured conversation within 1 second using computational power of regular PC. Other variant of attack would require 2 seconds of captured conversation and several minutes of computation effort. [33]

V. System Security

A. Access Control

Every system can be seen as an evolving environment that reside at particular state at any single point in time. As the system proceeds its inner states are changing accordingly. From IT security point of view it is not necessarily interesting to distinguish among all different states, but only among those that affect the protection of the system, called protection states. It is not interesting that value of some process variable changed from 1000 to 0, nevertheless, this fact gets important when this variable represents effective user ID of running process on Linux (process gained root privileges $\rightarrow$ protection state of the system changed). [1]

Access Control Matrix is the abstract construct that can precisely describe the protection state of the system. It represents the rights each active entity has over every other entity in the system. Active entities are referred to as subjects (example is running process) and all entities are referred to as objects (those are all passive entities like files and also all subjects are objects themselves). Access Control Matrix on the system with amount of subjects $|S|$ and amount of objects $|O|$ is represented as matrix A with dimension $|S| \times |O|$. Rights that subject s has over object o are recorded in the matrix cell $A[s, o]$. [1] Table I shows an example of access control matrix of a protection state of a system. It can be clearly seen that in a given protection system process 1 has rights r, w, o over file 1 while process 2 has only right r over the same file. In given example rights correspond to the well-known rights from UNIX systems (read, write, execute and own), but it is up to the definition which rights can be used in access control matrix.

	File 1	Process 1	Process 2
Process 1	r, w, o	e, o	e
Process 2	r		e, o

TABLE I: Access Control Matrix example

It is clear that each secure system needs to store the access control matrix in some form in order to know what rights subjects have over objects. Moreover, the system needs to provide access control mechanisms enforcing that subjects can exercise only those rights over objects that are assigned to them.

Decisions about how to implement the access control matrix model in the real system brings an interesting implications. Real world system is expected to accommodate hundreds of thousands of objects and thousands of objects, so storing the access control matrix in the memory as two-dimensional array is clearly not the best design. Moreover, most of the cells of such an access control matrix would be empty. [1]

I am going to discuss two fundamentally different approaches of implementation of access control mechanisms:

- Access Control Lists (ACL)
- Capabilities

Access control lists are implemented as a list stored with every object on the filesystem. This list enumerates all subjects and captures which rights the given subject has over the object (in other words access control list is column of access control matrix of the corresponding object). [1] When subject requests access to the object, system checks the access control list of a given object and grants or denies access based on its content.

Most of the common systems like UNIX and Windows are using the ACL approach of implementing access control mechanisms. ACL approach was shown to be susceptible to the "Confused Deputy Problem". It was presented by Norm Hardy in 1988 in his paper "The Confused Deputy:(or why capabilities might have been invented)". [36] Paper presents an example of system using ACLs as an access control mechanism with compiler `FORT` stored in folder `/SYSX`. Users of the system were charged for each use of this compiler. So everytime the compiler run it wrote billing information into `/SYSX/BILL` and some statistics about compilation into `/SYSX/STAT`. `/SYSX` folder was write-protected for ordinary user, but administrator granted compiler the right to write into that folder in order to store the billing information and statistics. When user invoked the compiler he could provide the path to file where he wanted the compiler to store debugging information. [36]

This is where the problem starts, if user provides `/SYSX/BILL` as a path to the file where to store debugging information, compiler would rewrite the billing information. It is important to stress out that the user itself does not have a right to write into `/SYSX/BILL`. The problem stems out from the fact that the compiler acts on behalf of two principals. It gets the right to write into `/SYSX` from administrator and is executed on behalf of ordinary user, thus allowing the ordinary user to write into the folder through the execution of the compiler. [36]

This problem can be elegantly solved by the means of capability based access control mechanisms. Capability based systems stores information about rights subjects have over objects within the subject. Every subject holds so called capabilities, which could be seen as single tokens defining the object and the right. If given subject wants to access the object it needs to provide the system with corresponding capability, otherwise the access is denied. If the system hosting the compiler uses capabilities, compiler would simply write into `/SYSX/STAT` and `/SYSX/BILL` using the capabilities granted to it by administrator and subsequently it would use the capability passed to it by invoker to write the debugging info. If the invoker is ordinary user he cannot possess the capability to write into any file in `/STAT` folder. He is therefore unable to rewrite any files there as in the case of ACL implementation and the confused deputy problem is mitigated. [36]

Naturally, access control matrix model requires somebody to set up which subjects have which rights over objects. There are several approaches how to tackle this problem generally falling into one of the following categories: [1] [37]

- Discretionary Access Control (DAC)
- Mandatory Access Control (MAC)
- Role-Based Access Control (RBAC)

Discretionary Access Control leaves it to the discretion of the users of the system to set up access rights over the objects. Generally, those systems use the concept of own right. User who creates the object becomes its owner. Subject with owner right over the object is subsequently allowed to grant rights over given object to another subjects of the system. [1] Example of DAC in practise are access control mechanisms employed in Linux system. Mandatory Access Control on the other hand is based on the set of access control rules that are defined by the administrator of the system and are enforced by the system, meaning that user cannot change them. [1] Example of MAC in practise are access control mechanisms provided by Security-Enhanced Linux (SELinux — Linux kernel security module).

DAC and MAC approaches were considered sufficient in the past to fulfil the needs of all possible systems. DAC was generally considered appropriate for private businesses while MAC was used in multilevel security environment of government systems processing classified information. The study researching the access control policy needs of commercial and federal US entities revealed in 1995 that especially DAC is not a good fit for commercial sector systems. [38] It found out that model where users would be assigned rights according to the roles they represent within the organization would be more appropriate for some entities researched.

The spontaneous demand for access control based on roles lead researchers to formally define RBAC in paper "Role-Based Access Control Models" in 1995. It defines four general approaches to implement RBAC starting with least complex solution labeled as $RBAC_0$ until the most complex one labeled as $RBAC_3$. The fundamental difference is in the concepts those RBAC models incorporate. The simplest $RBAC_0$ is defined as access control system that contains set of users, roles, permissions and sessions. Security administrator defines roles based on different roles within the organization and assigns appropriate permissions to those roles based on the principle of least privilege. Example of such a role is database administrator or project manager. Users are subsequently assigned roles they are performing within a company. Concept allows users to be assigned multiple roles. Everytime user is active, some subset of his/her roles are assigned to the actual session. This allows for a better use of the principle of least privilege. For example database administrator would have database administrator role active only during the administration process. During the time he is working on something else like replying to emails, researching on the internet he can use different role with appropriate privileges. [37] Studies show that changes in assignment of rights to the roles are much more rare than changes in assignments of users to the roles, which allows for streamlined administration process of RBAC compared to DAC. [37] [38]

$RBAC_1$ model was defined as model containing the whole concept of $RBAC_0$ and adding the hierarchy of roles allowing one role to inherit permissions from another role. For example project manager can inherit all permissions from project member role and add other permissions that are not available to the ordinary project member role. $RBAC_2$ model contains again the whole concept of $RBAC_0$, but in comparison to $RBAC_1$ adds constraints on user-to-role and role-to-permission assignments. It incorporates constraints like that one user cannot have two roles from mutually exclusive set assigned to him. This allows to stick to the principle of separation of privileges. Example would be the case when two people need to cooperate in order to issue and pay the bill. One person can only issue bill or pay the bill, but not both which reduced the likelihood of single person committing fraud. Most sophisticated RBAC model presented in the paper labeled as $RBAC_4$ combines all of the above concepts together. [37]

Administration of RBAC in more complex environments can be also done on the principle of RBAC. There might be a role that is allowed to create and delete users as they are joining and leaving the organization. Other role might be in charge of assigning proper roles to the users as they are changing their roles in the organization and third role might define the mapping of permissions to the roles according to the organization needs. [37]

B. Trusted Computing

As already described in the section I, security policies define what is and what is not allowed in the system. Those policies need to be properly enforced by the security mechanisms. For instance access control explained in the previous section needs to have trustworthy mechanisms that are enforcing it in order to ensure the security of the system. U. S. Department of Defense standard "Trusted Computer System Evaluation Criteria" from 1985 (also known as Orange Book) defined the term "Trusted computing base" (further referred as TCB) as following: "Trusted Computing Base includes hardware, firmware, and software critical to protection and must be designed and implemented such that system elements excluded from it need not be trusted to maintain protection." [39]

This in the other words means that if any security mechanism which is part of TCB gets compromised the security of the whole system is breached. [1] TCB in general Linux

consists of hardware, Linux kernel, system binaries and configuration files and if any of those parts contain the flaw (for example some application security vulnerabilities that are covered in detail in section VI-A) the system can get into disallowed (nonsecure) state. It was already known in 1985 that the bigger the TCB is the more likely it gets that it contains some flaw through which system security can be compromised. [39]

There are two fundamental approaches in the operating system kernel design which affect the subsequent size of TCB. Kernel can be either designed as monolithic kernel or μ-kernel (microkernel). Monolithic kernel is large piece of software that includes services for IPC, thread management, memory allocation, device drivers, file systems and other functionalities. Example of monolithic kernel based systems are UNIX or Linux. According to article "Who's Writing Linux?" in IEEE spectrum full Linux kernel in version 3.10 (released on 30th June 2013 [40]) approached 17 millions lines of code. [41] In general not all codebase would be used, but it gives a good picture about the size of well-known monolithic kernel and how unlikely it is to get reasonable assurance that those millions lines of codes are bug-free. On the other hand the basic concept behind μ-kernel is that the functionality can be implemented inside the kernel space only if implementing it outside of the kernel space is impossible. [42]

Although the advantages of μ-kernels and effects making the TCB smaller has on increased system security (due to the easier assurance process) were known at least from 1985 [39], μ-kernels were generally not used and accepted at least until mid-nineties. Jochen Liedtke published several μ-kernel related papers during those times including the one named "On μ-kernel Construction" in 1995. [42] At the time being it was generally perceived that the inefficiency is inherent to μ-kernels due to frequent switching between kernel and user space. Liedtke presented arguments that this perception is based on wrong assumptions and implementation and presented a well-performing μ-kernel design.

Concepts described by Liedtke were implemented in 1993 in μ-kernel named L3. [43] This theoretical and practical work restored the interest in μ-kernels with other μ-kernels like L4, Fiasco or Pistachio being released throughout nineties and after the turn of the millenium. [44] Another important milestone in the history of system security was formal verification of seL4 μ-kernel. Implementation of seL4 μ-kernel is the first one that was formally proved to be correct, giving the highest possible assurance in its correct functionality. It was achieved in 2009 at NICTA, ICT research center in Australia. [45]

The evolution explained above brought us a trustworthy system kernel, but system and its TCB consists of more than just a kernel. On the top of that widespread commodity systems are nowadays still based on monolithic kernels and nothing seems to indicate that it would change in a near future. Moreover, they tend to offer poor application isolation meaning that compromise of one process can often lead to the compromise of other processes and the whole system. This shortcomings prevent hosting multiple services with different security demands on a single system. [46]

Researches at Stanford university tried to tackle those issues by presenting their platform called Terra in 2003. [46] Goal of Terra was not to come up with single high-assurance system, but to allow multiple systems with varying security requirements to coexist on a single machine. It is based on the concept of Trusted Virtual Machine Monitor (TVMM) which allows multiple isolated virtual machines to run on a single tamper-resistant, general-purpose hardware. Therefore it would be up to the designer to group applications with the same security requirements to the same virtual machines. Depending on the security requirements those virtual machines might contain everything ranging from minimalistic OS containing only little more than bootstrapping code and one very high assurance application running on the top of it to the OS like Linux with mail client, web browser and other applications. [46]

In general μ-kernels provides interesting possibility for the secure paravirtualization. They offer small codebase that allows for superior scrutiny during the assurance process, allowing for high-assurance in separation of virtual machines running on the top of it (seL4 μ-kernel was even formally verified [45]). On the other hand paravirtualization brings the necessity of modifying the existing systems to run on μ-kernel API rather than on HW directly. L^4Linux project of porting Linux kernel to run on top of L4 compatible μ-kernel like Fiasco released its first port in 2004. Project is still active and as of February 2015 the last version of Linux kernel modified to run on the top of L4 API is 3.16. [47] Paper describing Terra was released in 2003 and opted for use of virtualization instead of paravirtualization in order to achieve compatibility with existing systems at the time. To support the choice authors argue that TVMM is a simple program that can be implemented within tens of thousands lines of code and as a such can achieve reasonable high-assurance of flawlessness the same way as μ-kernels. [46]

Terra also defines the concept of closed-box VM meaning that TVMM protects the content of the VM from disclosure or from unallowed tampering from the outside. Terra adds three additional features to the regular concept of VM isolation. First of all, it is root secure meaning that nobody, not even platform administrator, is able to break confidentiality and integrity of closed-box VMs. Secondly, it defines attestation as a possibility for closed-box VM to cryptographically assure the user that its content is what is expected. Lastly, it provides trusted path between user and VM, meaning that authenticity of user and VM is verified and the confidentiality and integrity of communication is ensured. [46]

VI. Application Security

Application security covers all security aspects of programs running on top of the operating system. There is an overwhelming amount of programs in the wild providing vastly different levels of technical quality of code as well as different levels of trustworthiness of their creators. Subsequent sections will cover both of those aspects separately.

A. Technical Aspects of Application Security

Programs are being developed by human beings and as a such it is inevitable that some security vulnerabilities are inadvertently introduced throughout the process. There exist multiple types of technical security flaws that can be found in desktop or web-based applications including buffer overflow,

SQL injection, cross-site scripting or flaws in the logic of the application. This section focuses on the history of buffer overflows and protective countermeasures.

Buffer overflow vulnerability is present in a case programmer fails to validate the provided input data. Most prominent example would be a C program that reads user's input and stores it into a pre-allocated array of characters without checking whether input fits into the allocated space. If the length of provided input is longer than array that was allocated for it, program simply rewrites the memory that is right after the array. If properly exploited it would allow attacker to inject arbitrary machine instructions within program's address space and subsequently affect the value of program counter in a way that injected machine code gets executed. Thus, allowing attacker to take over the control of a program with all privileges it possesses.

One of the first examples of successful large-scale exploitation of buffer overflow vulnerability was release of so-called "Morris worm" on 2nd November 1988. It targetted VAX and SunOS machines connected to the Internet at the time. One of the spreading mechanisms was exploitation of buffer overflow vulnerability in finger service that allowed the worm to execute arbitrary code on the target machine. Worm was not causing any intentional harm to the attacked computers, but contained a flaw that allowed multiple infections of the same machine which finally lead to the denial-of-service conditions. [48] Creator of a worm, Robert Morris, Jr., a graduate student at Cornell, was convinced and sentenced to a fine of $10,050, 400 hours of community service and three years of probation. [49]

Executing arbitrary code through stack-based buffer overflow was comprehensively described by Aleph One in 49th issue of Phrack magazine (published in 1996) in article "Smashing The Stack For Fun And Profit". [50] Interest and research in buffer overflow vulnerabilities gradually increased and it yielded 9687 CVE overflow vulnerabilities that were registered since 1999. [51] Those facts lead the developers of affected compilers and operating systems to introduce some countermeasures that would not need to rely on program developers producing secure code. Protective countermeasures include marking stack and heap address space as not executable, which would prevent attackers from executing injected code. Other countermeasures include address space layout randomization which maps heap, stack and linked libraries to unpredictable random memory addresses within process address space which makes it harder for the attacker to pick correct address for a jump.

Being unable to inject code that can be subsequently executed lead security researches to investigate if existing executable code that is mapped into process address space can be exploited. First successful exploit was described in 1997 by Solar Designer and was named return-into-libc overflow exploit. It exploits the fact that nearly all C binaries have C standard library linked into its address space and that overflow vulnerability would allow to jump to the arbitrary address. Solar Designer provided proof of concept of exploit that can call arbitrary function from libc (or any other library that is linked into the address space). Standard C library contains function `system` which allows to execute arbitrary shell command. Therefore, attacker can take over the control of process without injecting machine code into it, which render non-executable protection of much less use than it was perceived. [52]

Nevertheless, employing this approach gives an attacker more limited possibilities compared to the standard code injection. Standard code injection allows use of arbitrary machine code, while return-to-libc attack needs to stick to the functions that are linked into address space of a process. Moreover, return-to-libc attack does not allow any conditional branching, so attacker needs to stick to sequential execution of those functions. [53] This issue was approached by Hovan Shacham in his paper "The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)" which was published later in 2007. It introduces an approach that allows attacker to construct arbitrary machine code out of the code that is present in process address space. [53]

Main idea behind this technique is to use small code snippets that execute some actions and end with `ret` instruction (opcode `c3`). As author demonstrated, attackers does not need to depend solely on instructions intentionally used in the process. x86 instruction does not have a fixed length, meaning that if attacker sets the jump to the address that is not meant to be a beginning of some instruction, it is still very likely that the subsequent code will be interpreted as valid instructions by processor. Let's assume that our linked library contains following code snippet:

```
0:  48 83 ec 08         sub  $0x8,%rsp
4:  48 83 c4 08         add  $0x8,%rsp
8:  c3                  ret
```

If we jump to the address `0x0` code is executed as it is, however, if we jump to the address `0x1` instead, processor executes the following:

```
1:  83 ec 08            sub  $0x8,%esp
4:  48 83 c4 08         add  $0x8,%rsp
8:  c3                  ret
```

Author developed program that analyses the instructions contained in a library for all possible code snippets that can be used and explained how those snippets can be tied together in order to make process execute exactly what attacker wants it to execute, thus achieving similar outcome as in the case of injecting shellcode directly. [53]

B. Trust-based Aspects of Application Security

Previous section dealt with the resilience of applications against attacks that would make them behave in disallowed manner. However, there is also another aspect of application security that deals with trustworthiness of a given application. Due to the overwhelming size of software average company or individual need to use, it is out of the question to create all software in-house. As we need to rely on applications created by somebody else we also need to trust such applications that it does what it claims to and nothing else. Software that is performing some unwanted hidden operations or contain some hidden features that might do harm to a user are called trojan horses or backdoors. Trustworthiness of software we are using in our daily life have became very delicate problem mostly because of its huge amount and complexity that effectively

reduces the possibility to scrutinize properly every application before using it. Moreover, attempts of cybercriminals to take control of our computers and data in order to gain financial or other benefits and government agencies infamously spying on its own citizens further increase the complexity of what can and cannot be trusted.

Well-known example of sophisticated backdoor that can be introduced in a compiler was first outlined by Karger and Schell in their evaluation of security of Multics in 1974. [54] Attack was later implemented and presented by Ken Thompson in his Turing award lecture in 1984. The main idea behind the attack is to insert the trojan horse into the compiler that would affect the functionality of chosen programs after compilation, but it would be undetectable in compiler nor program source codes. It exploits the fact that C compiler itself is written in C. Therefore, if you want to add new functionalities into C compiler you need to "train" it how to compile those new features. Example given in Thompson paper shows how to include new escape sequence into the C compiler. If you want to add new escape sequence `\v` with ASCII value of 11 you first need to hardcode the value 11 into the new version of compiler and recompile it with the old one. Then value 11 can be replaced in the compiler source code by escape sequence `\v` and distributed. This way compiler knows that anytime it encounters escape sequence `\v` it should be translated to ASCII value 11 without any place in source code specifically mentioning value 11. [55]

This feature can be misused to "train" compiler to insert trojan horse into machine code of any program. Author implemented it a way that any time login program was compiled it included backdoor, so that login would accept valid password or predefined string. Therefore, allowing successful intruder to login as arbitrary user. Same pattern was included for compilation of compiler sources itself, so backdoor went undetected in source code of login program and compiler. Even though example of C compiler backdoor is already very complex to detect it is also possible to plant backdoor in lower levels (for example hardware microcode), which would be near to impossible to detect. All those facts emphasize the importance of trust we need to place in developers and distributors of applications and hardware. [55]

History saw several successful and unsuccessful attempts to introduce backdoors into widely used software solutions. Example of failed attempt happened in 2003 when unknown intruder tried to smuggle backdoor into Linux kernel. Linux kernel developers were using BitKeeper for revision control back then and all changes were being cloned into CVS repositories. Larry McVoy noticed on 5th November 2003 that there is a change in CVS that was done directly (it was not mirrored from BitKeeper). Most likely cause was that somebody broke into CVS server and tried to insert backdoor allowing for local privilege escalation. Following two lines of code were added: [56]

```
if ((options == (__WCLONE|__WALL)) &&
                    (current->uid = 0))
   retval = -EINVAL;
```

Second part of the condition contains assignment (=) instead of equality test (==), which means that whoever executes the system call with flags fulfilling the first half of the condition will be granted root privileges (user id 0). This attempt was considered as failed mainly because change only got into CVS mirror and was detected, so it could only affect people who pulled and used development version from CVS mirror. [56]

Another infamous example was a rootkit included on the audio CDs of Sony BMG in 2005. Rootkit existence was first revealed by Mark Russinovich in the blogpost "Sony, Rootkits and Digital Rights Management Gone Too Far" on 31st October 2005. It revealed that software that is part of audio CD and needs to be installed before one can listen to the music contains rootkit. Rootkit changed system calls the way it did not reveal existence of any files, folders, registry key or process which name starts with `$sys$`. Rootkit was active even when user was not listening to the music, it was very complicated to uninstall and this fact was withheld from EULA. [57] It was not until the huge wave of public discontent with those revelation spreaded around the world that Sony BMG started to act. Firstly, it temporarily halted production of its DRM scheme on 11th November and three days later it recalled infected CDs from retail stores and customers offering free replacement. [58]

Exploiting the trust of users is not only achieved by applications that are performing some covert operations unknown to the user, but also by tricking the user into taking some actions wanted by attacker or revealing his/her sensitive information including passwords, payment card details and others. This attack is known as phishing and in its most common form attacker attempts to trick victim by sending fake emails with spoofed address of origin. Message usually contains malware or asks victim to provide his/her personal information for some credibly-looking reason. If user follows the hyperlink provided in the message (very likely perfect copy of legitimate website) and enter his/her sensitive information it is captured and misused by attacker.

Phishing is younger technique how to exploit trust of the users compared to the distribution of rigged software. Invention of this technique dates back to mid-nineties and is connected to America Online (AOL) service. Attackers started to send out messages asking AOL subscribers to verify their accounts and confirm billing information in order to steal their account information. Later on, attackers turned their interest in more profitable ventures like stealing PayPal or payment card information. Thanks to the combination of uneducated users, improving phishing techniques and likely instant financial gains, phishing attacks were getting more and more common after the turn of the millenium. [59]

First survey examining which factors influence the success rates of phishing campaigns and why are users still fooled by attack with well-known pattern was conducted in 2006 by Rachna Djamila, J.D.Tygar and Marti Hearst. They asked 22 participants to evaluate 20 websites and to decide if those sites are legitimate or fake and commented the results in paper called "Why phishing works". [60] Authors discussed the influence that gender, education or previous computer skills might have on likelihood that individual correctly identifies counterfeit website. I personally believe that amount of participants and websites examined was too small to draw any valid statistical conclusions of such an influence. Nevertheless,

it was the first attempt to investigate reasons why phishing really works and it brought other interesting insights, especially about the inefficiency of countermeasures implemented in the browsers.

Among other facts it revealed that well-designed phishing site was able to fool 90 percent of participants. When it comes to the countermeasures that are supposed to inform users about possibility of counterfeit website it turned out that 15 out of 22 participants ignored warning about invalidity of website certificate. Moreover, almost quarter of them did not pay any attention at all on address bar, status bar and other security indicators that were present. Authors also noted that real-world results might be even worse than this because participants were encouraged to recognize phishing sites so they were more attentive. [60]

Therefore, it should not be too surprising that phishing attacks are still viable way of hacking into computer systems or stealing sensitive information. Some of the recent large-scale IT security incidents happened thanks to the ability of attackers to get initial access into the defense perimeter through phishing. One of such an examples is breach in Target Corp., US retail company from December 2013. Attackers managed to get in through phishing campaign affecting one of Target's contractors and they consequently compromised credit card and personal information of more than 110 million customers of Target company. [61]

VII. Present and Future of Information Security

Information technology is nowadays rapidly evolving and very innovative field of human knowledge. Technology and trends that were state-of-the-art yesterday are being replaced as obsolete today. Such progressively evolving environment is bringing great challenges in the field of information security. As already described in section IV there were examples in the past when rapid development of new information technologies left its security aspects improperly addressed. In my opinion, there are several new trends on the rise nowadays that require proper adjustments of security policies and mechanisms in order to ensure their viability and wide-adoption in short-term horizon among companies and individuals.

First and foremost, cloud computing technology draw a lot of attention since its inception in early 2000s as possible replacement for dedicated servers and mainframes. Cloud computing as a such has fundamental advantages that make it appealing. Its elasticity allows companies to dynamically pay cloud-computing providers exactly for resources they utilize without the need to acquire expensive hardware and human capital to operate it. [62] On the other hand, security challenges related with the fact that company stores their sensitive data in the public cloud outside of their control is one of the issues that needs to be addressed. Internet knows no borders, so using cloud services means storing your data in data center, that is very likely in the different country in completely different jurisdiction. Researches S. Subashini and V. Kavitha conducted a survey examining security issues of different cloud computing service delivery models and published its results in 2010. [63]

There are three different service delivery models for cloud computing: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). They differ in amount of abstraction client is offered by them. When SaaS is used provider maintains all the infrastructure from hardware, through networking and operating system up to the application level and client is presented with application that can be used for example through web browser. When utilizing this model, all security measures has to be taken by provider, so client has only extremely limited insight into the underlying infrastructure. On the contrary, IaaS model provides client with virtual server where client is able to control everything from operating system up. Thus, security aspects from system level upwards are in the hands of a client. PaaS is somewhere in between IaaS and SaaS, providing clients with control over the applications running in the cloud, but not operating system. Delivery model used thus determines how much control clients have over their data. Nevertheless, utilizing any aforementioned model still means that clients are required to voluntarily give up part of control over their own data to the cloud provider. [63] This clearly requires great deal of trust that provider puts in place appropriate measures to protect client's data and that those data are not deliberately accessed by some third party.

Such a trust is uneasy to obtain in connection with recent leaks of information about National Security Agancy (NSA) and other intelligence agencies surveillance programs. One of the leaked documents dated to April 2013 revealed existence of NSA program — PRISM. It described that NSA can directly access customer's data including emails, photos and other files that are stored in servers of major US cloud-service providers. Companies involved are among others Google, Microsoft, Facebook and Apple. [64]

There have been also incidents of cloud-solution provider failing to properly protect customer's data. Code Spaces, company providing project management tools and code repositories like SVN and Git was subject to the destructive cyber attack in June 2014. Service was build on the top of Amazon Web Services (AWS) IaaS solution and provided its services to customers as SaaS. Before the attack Code Spaces publicly displayed phrases like "Code Spaces has a full recovery plan that has been proven to work and is, in fact, practiced." on their website. Yet, attackers managed to get into their AWS account and asked for a ransom. When attackers found out that company was trying to regain full control over their AWS account they deleted all content hosted there. As Code Spaces failed to keep backup anywhere else it effectively lost all data of their customers and was forced to quit the business. [65]

Another noticeable trend in information technology is known as "Internet of Things". It represent the increasing numbers and varieties of devices that are being connected to the Internet. Everything ranging from your watch, refrigerator or home heating system through medical equipment to industrial installation can be connected to the Internet in order to increase its functionality. Such a development will obviously have profound security implications. While it is indeed bad when personal data of whole lot of people are compromised during a cyber attack, it is completely different story when attackers manage to affect physical operation of electrical grid, your heating system or medical equipment. Utilizing advantages of modern technologies in this field would require us to come up with security solutions that can scale well and provide appropriate levels of protection.

After all, every single aspect of our life and society is getting more and more dependent on modern technology. If we fail to properly secure digital devices that are now making our life simpler they might as well make our life much tougher in the future.

REFERENCES

[1] M. Bishop, *Computer Security: Art and Science*. Addison-Wesley, 2003.

[2] ——, "What is computer security?" *Security & Privacy, IEEE*, vol. 1, no. 1, pp. 67–69, 2003.

[3] F. W. Winterbotham, *The ultra secret*. Wiley-Blackwell, 1975.

[4] W. Diffie and M. E. Hellman, "New directions in cryptography," *Information Theory, IEEE Transactions on*, vol. 22, no. 6, pp. 644–654, 1976.

[5] "Archived fips publications," http://csrc.nist.gov/publications/PubsFIPSArch.html, accessed: 10.2.2015.

[6] FIPS PUB, "46, data encryption standard (des), national institute of standards and technology, us department of commerce (october 1999)," *Link in: http://http://csrc.nist.gov/publications/fips/archive/fips46-3/fips46-3.pdf*.

[7] ——, "197, advanced encryption standard (aes), national institute of standards and technology, us department of commerce (november 2001)," *Link in: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf*.

[8] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. ACM, 1987, pp. 1–6.

[9] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[10] O. Kömmerling and M. G. Kuhn, "Design principles for tamper-resistant smartcard processors," in *USENIX workshop on Smartcard Technology*, vol. 12, 1999, pp. 9–20.

[11] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology—CRYPTO'99*. Springer, 1999, pp. 388–397.

[12] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Advances in Cryptology—CRYPTO'96*. Springer, 1996, pp. 104–113.

[13] D. X. Song, D. Wagner, and X. Tian, "Timing analysis of keystrokes and timing attacks on ssh." in *USENIX Security Symposium*, vol. 2001, 2001.

[14] D. Brumley and D. Boneh, "Remote timing attacks are practical," *Computer Networks*, vol. 48, no. 5, pp. 701–716, 2005.

[15] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, "Silicon physical random functions," in *Proceedings of the 9th ACM conference on Computer and communications security*. ACM, 2002, pp. 148–160.

[16] R. Maes and I. Verbauwhede, "Physically unclonable functions: A study on the state of the art and future research directions," in *Towards Hardware-Intrinsic Security*. Springer, 2010, pp. 3–37.

[17] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010, pp. 237–249.

[18] C. Helfmeier, C. Boit, D. Nedospasov, and J.-P. Seifert, "Cloning physically unclonable functions," in *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 1–6.

[19] "Verayo – simply secure," http://verayo.com/, accessed: 11.2.2015.

[20] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. Wolff, "A brief history of the internet," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 5, pp. 22–31, 2009.

[21] "Rfc index," https://tools.ietf.org/rfc/index, accessed: 20.2.2015.

[22] G. A. Champine, D. E. Geer, and W. N. Ruh, "Project athena as a distributed computer system," *Computer*, vol. 23, no. 9, pp. 40–51, 1990.

[23] B. C. Neuman and T. Ts'o, "Kerberos: An authentication service for computer networks," *Communications Magazine, IEEE*, vol. 32, no. 9, pp. 33–38, 1994.

[24] D. J. Barrett and R. E. Silverman, *SSH, the Secure Shell: the definitive guide*. "O'Reilly Media, Inc.", 2001.

[25] IBM, "The secure sockets layer and transport layer security," http://www.ibm.com/developerworks/library/ws-ssl-security/#toggle, accessed: 28.3.2015.

[26] B. Möller, T. Duong, and K. Kotowicz, "This poodle bites: Exploiting the ssl 3.0 fallback," 2014.

[27] "15 years of wi-fi," http://www.wi-fi.org/discover-wi-fi/15-years-of-wi-fi, accessed: 29.3.2015.

[28] A. Stubblefield, J. Ioannidis, and A. D. Rubin, "A key recovery attack on the 802.11 b wired equivalent privacy protocol (wep)," *ACM transactions on information and system security (TISSEC)*, vol. 7, no. 2, pp. 319–332, 2004.

[29] "Aircrack-ng," http://www.aircrack-ng.org/, accessed: 29.3.2015.

[30] J. Vijayan, "Tjx data breach: At 45.6m card numbers, it's the biggest ever," http://www.computerworld.com/article/2544306/security0/tjx-data-breach--at-45-6m-card-numbers--it-s-the-biggest-ever.html, accessed: 29.3.2015.

[31] E. Chickowski, "Tjx: Anatomy of a massive breach," http://www.baselinemag.com/c/a/Security/TJX-Anatomy-of-a-Massive-Breach, accessed: 29.3.2015.

[32] GSMA, "Brief history of gsm & the gsma," http://www.gsma.com/aboutus/history, accessed: 29.3.2015.

[33] A. Biryukov, A. Shamir, and D. Wagner, "Real time cryptanalysis of a5/1 on a pc," in *Fast Software Encryption*. Springer, 2001, pp. 1–18.

[34] L. Green, I. Goldberg, and D. Wagner, "A pedagogical implementation of a5/1," http://www.scard.org/gsm/a51.html, accessed: 29.3.2015.

[35] I. Goldberg, D. Wagner, and L. Green, "The real-time cryptanalysis of a5/2," *Rump session of Crypto*, vol. 99, pp. 239–255, 1999.

[36] N. Hardy, "The confused deputy:(or why capabilities might have been invented)," *ACM SIGOPS Operating Systems Review*, vol. 22, no. 4, pp. 36–38, 1988.

[37] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[38] M. D. M. Gilbert, "An examination of federal and commercial access control policy needs," in *National Computer Security Conference, 1993 (16th) Proceedings: Information Systems Security: User Choices*. DIANE Publishing, 1995, p. 107.

[39] U.S. Department of Defense, "Trusted computer system evaluation criteria," *DOD 5200.28-STD*, 1985.

[40] "Active kernel releases," https://www.kernel.org/category/releases.html, accessed: 19.2.2015.

[41] "Who's writing linux?" http://spectrum.ieee.org/computing/software/whos-writing-linux, accessed: 19.2.2015.

[42] J. Liedtke, *On micro-kernel construction*. ACM, 1995, vol. 29, no. 5.

[43] ——, "Improving ipc by kernel design," in *ACM SIGOPS Operating Systems Review*, vol. 27, no. 5. ACM, 1994, pp. 175–188.

[44] "The l4 μ-kernel family," http://os.inf.tu-dresden.de/L4/, accessed: 20.2.2015.

[45] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish *et al.*, "sel4: Formal verification of an os kernel," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, 2009, pp. 207–220.

[46] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, "Terra: A virtual machine-based platform for trusted computing," in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. ACM, 2003, pp. 193–206.

[47] "Welcome to l4linux!" http://os.inf.tu-dresden.de/L4/LinuxOnL4/, accessed: 20.2.2015.

[48] M. W. Eichin and J. A. Rochlis, "With microscope and tweezers: An analysis of the internet virus of november 1988," in *Security and Privacy, 1989. Proceedings., 1989 IEEE Symposium on*. IEEE, 1989, pp. 326–343.

[49] "The robert morris internet worm," http://groups.csail.mit.edu/mac/classes/6.805/articles/morris-worm.html, accessed: 26.3.2015.

[50] A. One, "Smashing the stack for fun and profit," http://phrack.com/issues/49/14.html#article, accessed: 26.3.2015.

[51] "Cve details," http://www.cvedetails.com/vulnerabilities-by-types.php, accessed: 26.3.2015.

[52] S. Designer, "Getting around non-executable stack (and fix)," http://seclists.org/bugtraq/1997/Aug/63, accessed: 26.3.2015.

[53] H. Shacham, "The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)," in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 552–561.

[54] P. A. Karger and R. R. Schell, "Multics security evaluation: Vulnerability analysis," in *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*. IEEE, 2002, pp. 127–146.

[55] K. Thompson, "Reflections on trusting trust," *Communications of the ACM*, vol. 27, no. 8, pp. 761–763, 1984.

[56] L. McVoy, "Lkml: Bk2cvs problem," https://lkml.org/lkml/2003/11/5/121, accessed: 27.3.2015.

[57] M. Russinovich, "Sony, rootkits and digital rights management gone too far," http://blogs.technet.com/b/markrussinovich/archive/2005/10/31/sony-rootkits-and-digital-rights-management-gone-too-far.aspx, accessed: 27.3.2015.

[58] B. Schneier, "Sony's drm rootkit: The real story," https://www.schneier.com/blog/archives/2005/11/sonys_drm_rootk.html, accessed: 27.3.2015.

[59] "History of phishing," http://www.phishing.org/history-of-phishing/, accessed: 27.3.2015.

[60] R. Dhamija, J. D. Tygar, and M. Hearst, "Why phishing works," in *Proceedings of the SIGCHI conference on Human Factors in computing systems*. ACM, 2006, pp. 581–590.

[61] B. Krebs, "Email attack on vendor set up breach at target," http://krebsonsecurity.com/2014/02/email-attack-on-vendor-set-up-breach-at-target/#more-24313, accessed: 27.3.2015.

[62] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[63] S. Subashini and V. Kavitha, "A survey on security issues in service delivery models of cloud computing," *Journal of network and computer applications*, vol. 34, no. 1, pp. 1–11, 2011.

[64] "Nsa prism program slides," http://www.theguardian.com/world/interactive/2013/nov/01/prism-slides-nsa-document, accessed: 30.3.2015.

[65] "Aws console breach leads to demise of service with "proven" backup plan," http://arstechnica.com/security/2014/06/aws-console-breach-leads-to-demise-of-service-with-proven-backup-plan/, accessed: 30.3.2015.