Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/cor

Mobile device and malware related to this device

(2013)

Abstract: Next to the aforementioned stealing of private data, malware could also contain routines to capture voice calls and to silently record any conversations which are in range of the built-in microphone [53]. Depending on the privileges the malware has, this might happen completely in the background and will only be detectable by sophisticated monitoring of the whole operating system or the generated communication data. This type if eavesdropping is on a different layer than the aforementioned type in Section V.

Financially Motivated Attackers: As already shown in several papers [54]–[56], the business around malware got highly financially motivated in the recent years. Shady businesses were built to generate a lot of money from (initially) unaware victims. Not surprisingly, this trend has already reached smartphone malware as there is a strong connection between the smartphone and the MNO through some contract between the user and the provider in order to use the supplied services. Albeit the payment is often done in a flat rate model, some premium services are typically charged separately (*e.g.*, phone calls to special numbers or sending of short messages to some special services). The abuse of these services is ideal for attackers to generate money, *e.g.*, through offering a highly charged service number for short messages. An infected mobile device could covertly send messages to this number until the user might be aware of this situation on his monthly bill. One such malware is the malware *Trojan-SMS.AndroidOS.FakePlayer*, which pretends to be a movie player, but secretly sends messages to a service number which are highly charged [57]. Another way to generate money is to secretly redirect outgoing calls through a provider that generate an additional charge. Of course, this kind of MITM attack enables eavesdropping of the affected calls. Proof-of-concept malware with this behavior is evaluated by Liu *et al.* [58]. A third way to strip the victim of his money is to blackmail him. Although not yet seen on smartphones, one possibility is the encryption of private files and the release of the used key after some money has been paid ("ransomware"). An example of this is the *Gpcode*-Trojan for desktop computers. Mobile malware could set up similar extortion schemes, *e.g.*, by disabling certain services (such as email sending) on the mobile device and only re-enable them (for a short amount of time) after some payment has occurred (*e.g.*, by sending a premium short message to a number under the attacker's control).

Introduction

An important prospective question is the way criminals earn money with mobile devices. Currently, premium-rate services or foreign country calls are such a method. In the future, smartphone-based payment systems could be exploited as well. With an abuse database and by e
this abuse database on a mobile device, we expect
the currently working methods to cease.

The challenge for mobile network operators is contributing to the cessation of the current potential to earn money with exploiting mobile device security vulnerabilities, especially concerning premium services. Another challenge for future research in mobile device security is identifying the kind of successful attacks that cannot be solved with the security entities that are presented in this paper.

Mobile Botnets: Infected mobile devices are ideal remote controlled "machines", *e.g.*, within an established botnet. The different communication channels offered by smartphones enable much more (subtle) ways to control these machines next to the traditional, IP-based control structure of desktop malware. In addition, many smartphones are always turned on in contrast to ordinary computers. Singh *et al.* evaluated Bluetooth as the main command-and-control infrastructure [59] and Zeng *et al.* focused on the short message service [60]. Liu *et al.* showed that even a very small percentage of remote controlled mobile devices may successfully perform a DDoS attack on 911 call-centers [58]. In general, the topic on botnets is out of scope of this paper, but comprehensive literature exists [61], [62].

DoS Attacks Against Mobile Devices: Since mobile devices are battery powered, a huge power consumption can rapidly lead to the depletion of its power source. This can be easily done by malware by using all available CPU cycles for (junk) computations. A far more severe way to disable the service of a mobile device is the deletion or corruption of essential data stored at difficult to reach locations such as the E^2PROM. Fixing these issues is complex and often even impossible for average users because repairing requires fundamental knowledge of the device and can therefore often only be done by the manufacturer himself.

B. *SMS Vulnerabilities*

An incident of the early times of mobile phones (not even smartphones at that time) was an implementation bug in the SMS parser of the Siemens S55: receiving a short message with Chinese characters lead to a Denial of Service (DoS) [63]. This bug required a local firmware update, forcing the users to bring or send their device to customer service. This class is expected to be of less importance in the future, because modern smartphone architectures are increasingly allowing local or remote firmware updates, cf. Section II-C2.

A recent DoS attack is the "curse of silence" short message, which was published in late 2008 [64]. It is caused by an omitted sanity check of input data. Nokia published a removal tool one month after the attack was made public.

C. *MMS Vulnerabilities*

In 2006, a remote code execution exploit for mobile
using MMS as the attack vector was published by
r [65]. It exploited a buffer overflow in the MMS handling program of Windows Mobile CE 4.2. Being the

first of its kind, it supported the public fear of that time that mobile devices would start to become commonly attacked. The exploit received some attention by a technical audience and the MNOs, who published patches for affected devices. Anti-virus companies added the exploit to their signature databases, but the exploit never appeared as part of mobile malware and thus not much harm was caused by it.

There are two possible explanations for this. The first is the probability of succeeding with the message because it has to be guessed which memory slot is in use. A second and more probable explanation is the actual code base of the devices affected. In particular, Windows CE 4.2 was already succeeded by Windows CE 5 at the time when the exploit was published. Therefore, this vulnerability only affected comparatively outdated devices.

D. Mobile Web Browser

Mobile web browsers are an emerging attack vector for mobile devices. Just as common web browsers, mobile web browsers are extended from pure web browsing software to complete application frameworks with widgets or completely browser-based mobile devices We can expect that even security-relevant functions of the operating system will be accessible in the near future.

Industry requirements even include these security-relevant features. One example are the browser requirements of OMTP (Open Mobile Terminal Platform) [66]. More specifically, BR-2540 requires: "The browser MUST support the making of voice calls and video calls from a URI/IRI". BR-2570 suggests appropriate security mechanisms in the implementation of this requirement as follows: "The browser SHOULD ask for user confirmation before initiating any call from a hyperlink". Certain versions of the iPhone web browser complies with this requirement, but they enable browser-based dialers to create costs for the user without necessarily asking for confirmation.

Therefore, the mobile web browser as an application framework of its own is able to undermine the mobile device's security model: the original (and possibly secure) model of signed applications is replaced by the security model of the web browser developer. Examples for successful attacks—besides DoS attacks on the mobile Internet Explorer [67]—are the jailbreak of the iPhone, hacking the Android browser, and using the iPhone browser as a dialer.

E. Mobile Malicious Software

Investigating the damage potential of mobile malicious software is challenging today because this new kind of malware has the potential to undermine the trust of mobile phone users in their mobile telephony system as such. Therefore, we see the main research tasks for mobile device security in the attacks that can be committed by mobile malicious software. Here the mobile device can be seen as exhibiting arbitrary and possibly malicious behavior. We will first present preliminary work and then introduce malware detection mechanisms.

1) Surveys of Mobile Malware: This section provides in chronological order an overview of important surveys of mobile malware. Peikari presents an overview of Windows Mobile and Symbian OS malware [68]. An extensive article covering nearly all malware as of the time of its writing was given by Shevchenko [69]. A book by Eren and Detken lists known malware samples until 2006, surveys the weaknesses of mobile operating systems, and describes much of the mobile and the mobile device security knowledge of that time [70]. Tyssy and Helenius list infection routes and some examples of malware of the year 2006, but their focus is on countermeasures and media perception of mobile malware [71]. Bontchev notes mobile malware classification problems and chooses Symbian OS malware as an example [72]. Although not explicitly stated, his findings can be generalized for malicious software on any operating system. A survey of mobile malware is presented by Hypponen [5]. Besides a summary of mobile device security knowledge of that time, it shows in an illustrative comic cartoon that many repetitions of an installation attempt (via Bluetooth) could even break down the resistance of a security-conscious user. McAfee published a study in 2007 as a result of surveying mobile network operators [73]. This survey shows how MNOs start preparing defenses against mobile malware. The most recent work on this topic as of the time of writing is by Oberheide and Jahanian [8].

2) Malware Detection on Smartphones: Malware detection on smartphones is a difficult task. Although in principle not different from malware detection on desktop computers, the limited processing power of such devices poses a huge challenge. We already outlined this in Section II-C3 and now discuss several different malware detection strategies.

Signature Based Detection: This is the classic approach when a malware is identified and its characteristics are known. A signature may be generated and can thereafter be used to detect this special type. Classical AV software is signature based and works exactly this way on almost all computers. However, we cannot simply port this approach to smartphones. The main reason is that the matching algorithm must be regularly active to scan all processes for suspicious code. Obviously, this puts a heavy burden on the CPU and might even be noticeable by the user (*e.g.*, unresponsive graphical interface or a faster battery exhaustion). To avoid this, Oberheide *et al.* presented a virus scanner for mobile devices which offloads the actual scanning to the cloud [74]. Furthermore, the experience on desktop computers shows that signature based approaches are doomed to fail given the large number of newly emerging threats.

Next to the "classical signatures" for AV scanners, *static function call analysis* may provide clues about the intents of the corresponding program. This is typically done once at installation time for new programs. The used function calls may be classified and, if necessary, appropriate actions can be taken. This has been tested for the Android [75] and the Symbian platform [76].

A proactive way to detect malware before it even gets the chance to perform its malice intent is the way how Apple's *App Store* application vetting works. Each application that is uploaded by its developers is checked before it can be downloaded. Albeit the performed checks are unknown, its aim is to detect malicious code and to withhold the application if something suspicious is found. Since it is hard to detect malicious code hidden somewhere deep in the code path, some unwanted software slips through this mechanism from time to time [15], [77].

Anomaly Detection: In contrast to signature based detection approaches, anomaly detection techniques attempt to detect malware with unknown behavior. One example is *SmartSiren*, a tool developed by Cheng *et al.* [78]. SmartSiren is able to detect unknown malware based on its communication behavior through Bluetooth and SMS. Privacy conform data about these communication media is sent to a central proxy which attempts to detect abnormal behavior. This approach is able to detect fast spreading worms and "slow working" malware, which collects private data and forwards it from time to time in aggregated form.

Another example is the infrastructure presented by Portokalidis *et al.* [79]. Here a "tracer" runs on the mobile device and records all necessary information which is needed by a "replayer" to playback the instruction trace on an external virtual machine which represents a replica of the mobile device. This approach also offloads expensive computation to much more powerful computers in the cloud. This way, off-site virus detection routines may be applied in addition to complex, dynamic taint analysis, which may detect events such as buffer overflows or foreign code execution.

A completely different approach is evaluated by Liu *et al.* [80] and Kim *et al.* [81]. Since mobile devices have comparatively small batteries, malware should be detectable by the amount of battery power consumed by their conducted instructions. If the running applications, the user behavior, and the state of the battery is well known and precisely defined, additional hidden (malicious) activity can be detected. However, it is unclear and an open research question how well malware can be detected on smartphones that is in daily use, especially with continuously changing user behavior.

Rootkit Detection: Malware with high privileges may attempt to hide itself at kernel level. The rootkit techniques do not differ from ordinary computers and, hence, their detection is to a certain extent identical—and therefore very hard. A first rootkit for Android has already been presented [82] and Bickford *et al.* evaluated rootkit detection on mobile devices [83]. It is an open question how rootkits on smartphones can be detected effectively and efficiently.

Software-based Attestation: Jakobsson and Johansson describe an approach to retroactively detect any active software based on *memory printing* [84]. The idea is to use light-weight cryptographic constructions with the property that it takes notably longer to compute a given function when the performing algorithm is given less usable RAM than for which it was configured. This approach is suited to detect software that wants to hide its presence on mobile devices. Active malware, whose memory is swapped out to much slower (Flash) memory during execution, will experience a huge latency penalty which is measurable. This makes active malware detectable through either observed memory changes or especially due to timing discrepancies when the malware attempts to evade detection during attestation by computing the expected result for the attestation.

F. Operating System Protection

Because smartphones deal with broader and broader application domains, their operating system and their programs become comparable to desktop computers. Both systems share the same architecture and in many cases even the exact same technologies, *e.g.*, the operating system or web browser back end. Thus, their security can be tightened with the same technologies. We now focus on some ways to enhance the security of smartphone operating systems.

Limited privileges and process isolation: Exploited applications may run foreign code within the boundaries of their given privileges. Higher privileges endanger the whole system and are often not necessary for the vast majority of applications. Smartphone applications should be run with the same principles in mind as their counterparts in ordinary computers. A good example is the Android platform, where applications run with different UIDs and are further separated through their own JVMs. Virtualization in general may enhance the overall security of smartphones (*e.g.*, separated VMs for private and work-related tasks), but currently there is no (hardware) support for this yet.

Hardened Kernels: Ordinary computer operating system kernels and utilities constantly raise the stakes to execute foreign code through critical security bugs. These techniques should also be used at the heart of smartphone operating systems and include *Address Space Layout Randomization*, *stack protection* and *non-executable writable memory*. *Mandatory Access Control* lists may further enhance overall smartphone security. It is ongoing work to enable all these techniques on the different platforms.

Sane Default Settings: Smartphone services and applications should have sound default configurations and should only run when their services are required. One such example is Bluetooth connectivity. Another is shown by Habib *et al.* [85], where some Symbian based smartphones are prone to DoS attacks in their default configuration via a network service that is reachable by default. To our knowledge, such an evaluation has not been done except for Windows Mobile and Symbian platforms.

Updates: The more people work in the area of smartphone security, the more likely it is that (security) bugs will be found. In order to close the corresponding emerging security holes, applications and operating systems need to be updated. This has to been done in an easy and straightforward fashion, as common users are not interested nor motivated in long update procedures because of their missing security understanding, cf. Section VII. Better update procedures are an open research question. A challenge for future offensive research can be the abuse of firmware flashing functionality, which leads to more subtle attacks. However, note that this kind of attacks might be detected successfully.

Software Attestation: A feature of smartphones is the ability to install all kinds of (closed source) 3$^{\text{rd}}$-party software. Those applications may contain unwanted routines which are very hard to detect. One example is the previously mentioned private data stealing routine in a game. The Android OS allows to see which capabilities (Internet-access, send/receive SMS, ...) an application requests during install-time and to eventually deny them. Sadly, this is an all-or-nothing decision. Either the application is installed and may anytime make use of the granted capabilities, or it is not installed and therefore not usable. But even an unsuspicious looking weather application which requests forecasts over the Internet connection and which might automatically retrieve updates based on the current location (which is known through the embedded GPS sensor) is perfect to spy on the user. In order to detect such unwanted behavior, some research has been done on this topic. *Kirin* [86] is a framework for Android which decides on the basis of the stakeholders' *policy invariants* which application requests are granted or denied. Ongtang *et al.* developed *SAINT*, a modified infrastructure for Android which assigns permissions during install-time for run-time access to or communication between applications [87]. Another proposed Android security tool is *SCanDroid* [88], which may automatically detect unwanted information flows in applications based on the requested capabilities. The downside of this approach is that the source code is required and that it is solely for analysis purposes as no intervention is possible.

Complementary approaches are *TaintDroid* [89] and *PiOS* [90]. Both systems perform dynamic taint analysis and are able to reveal hidden and possible unwanted information flows of private data. These tools do not need the source code for their operation, but work on the binary level. A related issue is the ongoing challenge of application revocation of mobile device applications [15].

G. Limited Graphical User Interface

We will now look into two challenges concerning the user interface of mobile devices. First, it is possible that the user interface does not display the message that the program or the operating system intended to. Examples are APIs for dialog boxes that accept strings of arbitrary length for the message to be displayed. Niu *et al.* presented some phishing techniques based on these inadequatenesses [91]. Second, and even more challenging, is malware that is able to simulate user actions, *e.g.*, by automatically reacting to security confirmations. Some desktop computer operating systems provide APIs for this task and it can be imagined that mobile operating systems will provide this functionality in the future. It becomes even more intriguing in combination with malware that rewrites some parts of the OS.

A first solution to these problems is the introduction of Turing tests (CAPTCHAs) for every security-relevant event on the mobile device in order to prove that an event was confirmed by a human user. The task for future research could be to explore the portion of security problems that remain when this solution is applied. An additional question lies in the area of human computer interaction (HCI) design: enough to be of use, but not so much so that it becomes a burden.

Conclusion

Many studies have been performed to evaluate the security knowledge of the average user. Most of them show what already the well-known study of Whitten and Tygar [92] found out: normal users are not able to use security mechanisms correctly. Several attempts have been made to simplify the security interface for users, but even the very simple Windows security slider with only *four* possible positions was not completely understood and therefore wrongly set by users who rated themselves as *IT proficient* [93].

Therefore, we need to ask ourselves the purpose of these numerous security mechanisms if the user does not understand them. And even if he understands to work with one specific mechanism, another mechanism might be difficult to grasp for him. People working in security do want to achieve the desirable goal of more security-aware users, but for the vast majority of users, their will, their interest, and in particular the time they can devote to learn more than one security mechanism, is likely to be limited.

Security and usability started out of general research on HCI. We can trace it back to the famous study by Whitten and Tygar in 1999 [92] and observe that it gained attention in subsequent years [94], [95]. User studies regularly show that security mechanisms are neither understood nor correctly used by most of their users [92], [93], [96], [97]. In addition, some authors propose to embed security in products [98] and in the development process [99] rather than having it stand-alone. Usability heuristics have been developed by Nielsen [100] and Shneiderman and Plaisant [101]. They are a good starting point for usability of security solutions.

Making our scope a bit broader, we also have to see the administrator or security professional as a possible attack vector: new and changing environments, poorly documented hard- and software, and interplay of components which were not developed to work together.

References

[1] J. Campbell, “Speaker recognition: a tutorial,” *Proc. IEEE*, vol. 85, pp. 1437–1462, Sept. 1997.

[2] D. A. Reynolds, T. Quatieri, and R. Dunn, “Speaker verification using adapted Gaussian mixture models,” *Digital Signal Processing*, vol. 10, no. 1–3, pp. 19–41, 2000.

[3] D. A. Reynolds, “Comparison of background normalization methods for text-independent speaker verification,” in *Proc. Eurospeech*, 1997.

[4] D. A. Reynolds and R. C. Rose, “Robust text-independent speaker identification using Gaussian mixture speaker models,” *IEEE Trans. Speech Audio Processing*, vol. 3, no. 1, pp. 72–83, 1995.

[5] J. L. Gauvain and C.-H. Lee, “Maximum a posteriori estimation for multivariate Gaussian mixture observations of Markov chains,” *IEEE Trans. Speech Audio Processing*, vol. 2, pp. 291–298, Apr. 1994.

[6] E. Bocchieri, “Vector quantization for the efficient computation of continuous density likelihoods,” in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, 1993, pp. 692–695.

[7] K. M. Knill, M. J. F. Gales, and S. J. Young, “Use of Gaussian selection in large vocabulary continuous speech recognition using HMMs,” in *Proc. Int. Conf. Spoken Language Processing*, 1996.

[8] D. B. Paul, “An investigation of Gaussian shortlists,” in *Proc. Automatic Speech Recognition and Understanding Workshop*, 1999.

[9] T. Watanabe, K. Shinoda, K. Takagi, and K.-I. Iso, “High speed speech recognition using tree-structured probability density function,” in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, 1995.

[10] J. Simonin, L. Delphin-Poulat, and G. Damnati, “Gaussian density tree structure in a multi-Gaussian HMM-based speech recognition system,”

[11] T. J. Hanzen and A. K. Halberstadt, “Using aggregation to improve the performance of mixture Gaussian acoustic models,” in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, 1998.

[12] M. Padmanabhan, L. R. ahl, and D. Nahamoo, “Partitioning the feature space of a classifier with linear hyperplanes,” *IEEE Trans. Speech Audio*
in *Proc. Int. Conf. Spoken Language Processing*, 1998.
Processing, vol. 7, no. 3, pp. 282–288, 1999.

[13] R. Auckenthaler and J. Mason, “Gaussian selection applied to text-independent speaker verification,” in *Proc. A Speaker Odyssey—Speaker Recognition Workshop*, 2001.

[14] J. McLaughlin, D. Reynolds, and T. Gleason, “A study of computation speed-ups of the GMM-UBM speaker recognition system,” in *Proc. Eurospeech*, 1999.

[15] S. van Vuuren and H. Hermansky, “On the importance of components of the modulation spectrum of speaker verification,” in *Proc. Int. Conf. Spoken Language Processing*, 1998.

[16] B. L. Pellom and J. H. L. Hansen, “An efficient scoring algorithm for Gaussian mixture model based speaker identification,” *IEEE Signal Processing Lett.*, vol. 5, no. 11, pp. 281–284, 1998.

[17] J. Oglesby and J. S. Mason, “Optimization of neural models for speaker identification,” in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, 1990, pp. 261–264.

[18] Y. Bengio, R. De Mori, G. Flammia, and R. Kompe, “Global optimization of a neural network—hidden Markov model hybrid,” *IEEE Trans. Neural Networks*, vol. 3, no. 2, pp. 252–259, 1992.

[19] H. Bourlard and C. J. Wellekins, “Links between Markov models and multilayer perceptrons,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 12, pp. 1167–1178, Dec. 1990.

[20] J. Navrátil, U. V. Chaudhari, and G. N. Ramaswamy, “Speaker verification using target and background dependent linear transforms and multi-system fusion,” in *Proc. Eurospeech*, 2001.

[21] L. P. Heck, Y. Konig, M. K. Sonmez, and M. Weintraub, “Robustness to telephone handset distortion in speaker recognition by discriminative feature design,” *Speech Commun.*, vol. 31, pp. 181–192, 2000.

[22] A. Dempster, N. Laird, and D. Rubin, “Maximum likelihood from incomplete data via the EM algorithm,” *J. R. Statist. Soc.*, vol. 39, pp. 1–38, 1977.

[23] K. Shinoda and C. H. Lee, “A structural Bayes approach to speaker adaptation,” *IEEE Trans. Speech Audio Processing*, vol. 9, no. 3, pp. 276–287, 2001.

[24] K. Fukunaga, *Introduction to Statistical Pattern Recognition*. New York: Academic, 1990.

[25] J. C. Junqua, *Robust Speech Recogntion in Embedded Systems and PC*

[26] U. V. Chaudhari, J. Navrátil, S. H. Maes, and R. A. Gopinath, “Transformation enhanced multi-grained modeling for text-independent speaker recognition,” in *Proc. Int. Conf. Spoken Language Processing*, 2000.

[27] Q. Lin, E.-E. Jan, C. W. Che, D.-S. Yuk, and J. Flanagan, “Selective use of the speech spectrum and a VQGMM method for speaker identification,” in *Proc. Int. Conf. Spoken Language Processing*, 1996.

[28] S. Raudys, *Statistical and Neural Classifiers: An Integrated Approach to Design*. New York: Springer, 2001.

[29] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation,” in *Parallel Distributed Processing*. Cambridge, MA: MIT Press, 1986, pp. 318–364.

[30] [Online] Available: http://www.nist.gov/speech/tests/spk/index.htm.

[31] J. Pelecanos and S. Sridharan, “Feature warping for robust speaker verification,” in *Proc. A Speaker Odyssey—Speaker Recognition Workshop*, 2001.

[32] B. Xiang, U. V. Chaudhari, J. Navrátil, N. Ramaswamy, and R. A. Gopinath, “Short-time Gaussianization for robust speaker verification,” in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, 2002.

[33] G. R. Doddington, M. A. Przybocki, A. F. Martin, and D. A. Reynolds, “The NIST speaker recognition evaluation—overview, methodology, systems, results, perspective,” *Speech Communication*, vol. 31, pp. 225–254, 2000.

Bing Xiang (M’03) was born in 1973 in China. He received the B.S. degree in radio and electronics and M.E. degree in signal and information processing from Peking University in 1995 and 1998, respectively. In January, 2003, he received the Ph.D. degree in electrical engineering from Cornell University, Ithaca, NY.

From 1995 to 1998, he worked on speaker recognition and auditory modeling in National Laboratory on Machine Perception, Peking University. Then he entered Cornell University and worked on speaker recognition and speech recognition in DISCOVER Lab as a Research Assistant. He also worked in the Human Language Technology Department of IBM Thomas J. Watson Research Center as a summer intern in both 2000 and 2001. He was a selected remote member of the SuperSID Group in the 2002 Johns Hopkins CLSP summer workshop in which he worked on speaker verification with high-lelvel information. In January, 2003, he joined the Speech and Language Processing Department of BBN Technologies where he is presently a Senior Staff Consultant-Technology. His research interests include large vocabulary speech recognition, speaker recognition, speech synthesis, keyword spotting, neural networks and statistical pattern recognition.

Toby Berger (S’60–M’66–SM’74–F’78) was born in New York, NY, on September 4, 1940. He received the B.E. degree in electrical engineering from Yale University, New Haven, CT in 1962, and the M.S. and Ph.D. degrees in applied mathematics from Harvard University, Cambridge, MA in 1964 and 1966, respectively.

From 1962 to 1968 he was a Senior Scientist at Raytheon Company, Wayland, MA, specializing in communication theory, information theory, and coherent signal processing. In 1968 he joined the faculty of Cornell University, Ithaca, NY where he is presently the Irwin and Joan Jacobs Professor of Engineering. His research interests include information theory, random fields, communication networks, wireless communications, video compression, voice and signature compression and verification, neuroinformation theory, quantum information theory, and coherent signal processing. He is the author/co-author of Rate Distortion Theory: A Mathematical Basis for Data Compression, Digital Compression for Multimedia: Principles and Standards, and Information Measures for Discrete Random Fields.

Dr. Berger has served as editor-in-chief of the IEEE TRANSACTIONS ON INFORMATION THEORY and as president of the IEEE Information Theory