Hello there!

Thank you for your interest in Ruth’s musings in security. I aim to read one academic paper in cryptography (theory) and computer security (application) each week. Here are the musings from the week 11/11/2016 to 18/11/2016. I am very far from being an expert on these topics, therefore, if you need to contact me, to report errors in my stuff, my email address is thisemailisnottruthless@gmail.com. Also, for clarification, this is not research that was done this week. This is research that was read by Ruth this week =P

This Week in Crypto: “Practical Key Recovery on MANTIS5”

by Christopher Dobranig, Maria Eichlseder, Daniel Kales, Florian Mendel

Premise: MANTIS is a lightweight tweakable block cipher that was proposed by Beierle et al. at CRYPTO 2016, improving on PRINCE. The authors presented two versions of MANTIS, MANTIS5 and MANTIS7 which have 10 rounds and 14 rounds respectively. While the designers presented a proof against “practical attacks”, the authors present a key-recovery attack that violates this claim.

Description of MANTIS5: MANTIS has a 64-bit message block, a 64-bit tweak and a 128-bit key. The 64-bit values are mapped to 4 × 4 states of 4-bit cells. We will very briefly overview the structure of this cipher. The overall structure of the cipher is similar to that of PRINCE, which we present pictorially in the below image:

Here, the round function \( R_i \) and inverse round function \( \overline{R}_i \) are described pictorially here:

In these diagrams, \( k_0 || k_1 \) is the 128-bit key, while \( k'_0 \) is a derived key, where \( k'_0 = (k_0 >> 1) + (k_0 >> 63) \). \( h, P \) are cell permutations. Also, \( \alpha \) is a fixed constant, and the \( C_i \) are round constants. \( S \) is a 4-bit S-box and \( M \) is a mix-column operation using a fixed matrix. \( T \) is the 64-bit tweak.

Security Claims: The designers of MANTIS showed the security of the cipher against differential cryptanalysis by modeling the differential behavior as a mixed integer linear program. With this, they conclude that since the maximal differential probability of the S-box makes it such that MANTIS5 is secure against “practical attacks”, which they describe as related-tweak attacks with data complexity \( 2^d \) less than \( 2^{30} \) chosen plaintexts (or \( 2^{40} \) known plaintexts) and computational complexity at most \( 2^{126−d} \) block cipher calls, similar to the PRINCE challenge.

Differential Characteristic: The authors present an attack that violate this security claim by considering a family of differential characteristics, instead of a single fixed input and output difference. By clustering several related differential characteristics together, they obtained a much better probability.

Like the designers, the authors note that the optimal differential probability of \( 2^{-68} \) is obtained by input different \( a \) and output difference \( a \). However, they proceed by
relaxing some of the constraints by considering other differences that can be mapped to and from a by the the S-box $S$, namely $\{ 5, a, d, f \}$. Then, they consider separately the input and output differences they want to allow, out of that set, for each round to obtain better probabilities than the ones presented by the designers of MANTIS. With this, they achieve a family of characteristics with probability $2^{-40.51}$ up to round 9, with which one can develop a 30-bit filter for the output of round 10 to aid key recovery. This is a much better probability than $2^{-68}$ suggested by the designers.

The authors first analyze how this attack can be done within the stipulated data complexity, and its likelihood of success, given that data complexity. They show that, to run the attack once would require $2^{26}$ queries, with which we can generate $2^{41}$ possible plaintext-tweak pairs. From this, we expect that the number of valid pairs to be $2^{41-40.51} \approx 1.40$. With $2^{30}$ plaintexts, this can be repeated to derive $\approx 22.47$ pairs. This was corroborated by experimental results.

**Key Recovery:** The authors then describe how one can proceed with the key recovery. This starts with filtering the pairs by the expected differences in particular cells consistent with the differential characteristic. This will reduce the number of pairs from $2^{41}$ to $2^{19}$, without removing any of the valid pairs. This can be made done efficiently for $2^{30}$ state xor operations.

Then, the authors retrieve 44 bits of $k_0' + k_1$. They arrive at the number $2^{14}$ by noting that only in 11 of the 16 cells does the differential characteristic help us to check for a consistent key by inverting the results of round 10 on valid ciphertext pairs. Since we are still unsure which of the pairs are valid, they search for the partial keys consistent with any of the $2^{19}$ ciphertext pairs, and whittle the number down with 4 different initial structure from their family of differential characteristics. This can be done efficiently by guessing the bits in each column separately, requiring the storage of $2^{25}$ states, and the equivalent time of $2^{32.54}$ encryptions.

Next, they retrieve 32-bits of $k_0 + k_1$. After the previous steps, the authors are also able to reduce the number of ciphertext pairs down to the valid ones (with high probability). This can be used to derive 32-bits of the initial whitening key by searching for those consistent with the differential characteristics, in the cells that have a non-zero difference. This is done similar to the above step, in $2^{30.42}$ encryptions.

Finally, they retrieve all the bits of the keys $k_0, k_1$ using linear programming. The above provides 76 equations to the solver, and a further 14 can be derived from the knowledge of the above 76 bits, based on the differential characteristic chosen. Therefore, the remaining 38 bits were brute forced, through a guess-and-determine method, with the $2^{38}$ time complexity dominating all other steps.

**Proof of Concept:** The authors then present a proof of concept, a C/C++ code that successfully retrieved MANTIS$_2$ keys. This took 16 minutes to generate the $2^{19}$ candidate pairs, a further 40 mins to derive the 44 bit partial key, and the rest of the attack took 3 minutes to complete. Overall, the proof of concept took less than an hour, on a single core. The authors note that this is easily parallelizable as well.

**Conclusions:** The authors note that many of the weaknesses of MANTIS stems from its use of lightweight building blocks from the Midori block cipher, which does not have good security properties. Also, the Midori round function does not interact well with the PRINCE inspired structure, where the 4 inner layers can be considered a “superbox” in the establishment of a family of differential characteristics. Finally, its security analysis failed to consider all cases of differential cryptanalysis when it gave its security claims, which allowed the authors to violate the security claims of MANTIS.

The authors also note that these techniques can be easily adapted to MANTIS$_7$.

---

This Week in Security: "Android Root and its Providers: A Double-Edged Sword"

by HANG ZHANG, DONGSONG SHE, ZHIYUN QIAN

**Premise:** The authors address an interesting problem that is at the heart of most commercial technology today: that the user of a device does not have full control over it. More specifically, in the case of Android smartphones, that many users hope to bypass restrictions set by carriers, operating systems and hardware manufacturers. With full control over the device, the user can run paid Apps for free, or change carrier as they please. Users do this in two ways, one is a “hard” root, where the binary is flashed externally via an update package or ROM (i.e. modified android software is loaded), and the other is a “soft” root, making use of software that can be exploited. The authors focus on soft roots, studying how it is done and could potentially be exploited by malware authors to their own ends. They chose to do so because they believe the soft root is both more dangerous and more ubiquitous.

**Available root exploits:** The authors first attempt to exhaustively collect publicly known Android root exploits to understand their characteristics. They did this by trawling academic papers, research projects, published books, online knowledge bases and other Internet searches. They retrieved a list of 73 exploits or vulnerabilities. They note also that some exploits required multiple vulnerabilities, and some vulnerabilities were exploited in more than one exploit.

They then studied the impact of the root exploit, by considering its impact and coverage:

- **Linux Kernel:** Due to its privileged position, exploits naturally target the Linux kernel to gain control
over the whole divide. TowelRoot is an example of this.

- **Vendor-Specific Kernel or Drivers:** When vendors customize the kernel, or provide specific device drivers for various peripherals (e.g. the camera), this code runs inside the kernel space and can also lead to full control over the device. Attacks on this would be specific to the devices with similar customization.

- **Libraries Layer:** Exploits at this level target Android or external libraries that are on the phone to support different applications. Zergrush is an example of such an exploit. Depending on the library exploited, this can have far reaching consequences.

- **Application and Application Framework:** These exploit system applications and services. The danger of these vary depending on how commonly used these system applications are used.

Overall, the authors found 73 exploits, of which 54 are vendor-specific, and 19 were general exploits. The application layer had the largest number of exploits, although most were vendor specific. Studying the timeline of vulnerabilities and exploits, the authors noted that a large number of vulnerabilities were introduced in 2013 introduced by vendor customization, at the external drivers and applications layer.

Next, the authors commented on the availability of exploits, in the form of source code or binaries, to potential malware authors. Of the 73 exploits, the authors located 68 of them online, 46 in the form of source code and 22 with binaries available. They noted that while the source code would seem to be more useful to a malware author, these were often just a "proof of concept", and the binaries found were more robust. For example, TowelRoot is binary only, but has evolved over three major revisions, supporting different devices.

Next the authors comment on the requirements of the exploits. These were the major requirements that they found:

- **User Interaction:** There were a few which required a user to hit buttons, open Apps or perform other manual tasks.

- **adb shell through a PC connection:** In some exploits, adb shell connection is required to modify settings or write to files.

- **Reboot:** Many exploits require at least one reboot.

Next, they studied whether root exploits were easily detected by anti-virus software, since they can be potentially abused by malware authors. Therefore, they installed 21 root exploits in the form of apk files, or ARM ELF executables onto the phone as "Apps" (packaging them if need be). Installing one anti-virus software at a time, they test if the anti-virus will flag the apps. Only 1 was not flagged by the 4 antivirus softwares used, and 2 were detected by only one antivirus software. The authors do also note that certain antivirus softwares were able to flag even very recently discovered exploits, suggesting they are very sensitive to publicly available exploits. They also note that the exploits engineered by large root providers are surprisingly undetectable, compared to those written by individuals. However, when the authors took a handful of these in the form that they were "packed" by the exploit companies (not just the raw exploit), none of them were detectable, which is very worrying.

Finally, they studied the adaptability of these exploits. This refers to the number of different devices supported by each exploit. Often, exploits work only in specific environments, such as the type of CPU, kernel version and OS version. It is also believed that this discourages malware authors from using that exploit.

**Root Providers:** The authors then study the nature of root providers. They noted that there is a common architecture to the providers studied, with the service being provided either by a PC-side program and/or an Android one-click root App.

They also searched for the number of exploits offered by providers, providing a lower bound on that value in that way. Surprisingly, the one with the most goes to well over a hundred, much higher than public sources highlight.

Then they studied the protection strength of the products. Expectedly, larger root providers used strong protection on their products, while smaller ones employed little to no protection. Larger root providers made use of obfuscation, encryption and event tamper detection in their products, and smaller ones had tamper detection that was easy to bypass. However, the strong protection, in some cases, could be broken through some weaknesses. In fact, even the most difficult provider only took a graduate student (who was not a professional hacker) a month of part-time work to break.