Embedded Firmware Diversity for Smart Electric Meters

Stephen McLaughlin, Dmitry Podkuiko, Adam Delozier
Sergei Miadzvezhanka, and Patrick McDaniel
{smclaugh,podkuiko,delozier,swm5344,mcdaniel}@cse.psu.edu

Abstract

Smart meters are now being aggressively deployed worldwide, with tens of millions of meters in use today and hundreds of millions more to be deployed in the next few years. These low-cost ($\simeq$ \$50) embedded devices have not fared well under security analysis: experience has shown that the majority of current devices that have come under scrutiny can be exploited by unsophisticated attackers. The potential for large-scale attacks that target a single or a few vulnerabilities is thus very real. In this paper, we consider how diversity techniques can limit large-scale attacks on smart meters. We show how current meter designs do not possess the architectural features needed to support existing diversity approaches such as address space randomization. In response, we posit a new return address encryption technique suited to the computationally and resource limited smart meters. We conclude by considering analytically the effect of diversity on an attacker wishing to launch a large-scale attack, showing how a lightweight diversity scheme can force the time needed for a large compromise into the scale of years.

1 Introduction

The smart grid is now a reality. Millions of homes and businesses have been connected to regional and national networks that enable real time reporting and control of electrical use. This digitization of grid control systems offers substantial benefits for society; increased efficiencies and information availability can enable cheaper and greener energy generation, less loss in energy storage and transmission, better fault isolation and recovery, and support for widespread consumer use of alternative energy sources, e.g., consumer-generated wind and solar energy.

Smart meters are the consumer conduit to the grid. Replacing the near century-old electromechanical meters attached to the exteriors of many buildings, smart meters are enhanced sensors that record power use and act upon control signals from the grid. Smart meters are being aggressively deployed: in addition to substantial investments by industry, over \$4.3 billion was recently allocated to the deployment of the smart grid [23].

The move to digital grid control systems introduces concerns about their security [16, 20, 19]. The smart grid is a complex system of sensors, networks, and computing resources. Attacks against the smart grid networks and computing elements can range from fraud, to denial of service, to privacy loss [21].

Of particular concern are the smart meters themselves. Our and others' recent analyses have shown that current meters have exploitable flaws [22, 28]. Underlying these vulnerabilities is that fact that the security mechanisms they employ are sometimes naive or incomplete. This heightens fears of a "billion dollar bug" [21]. This hypothetical bug would exploit a common security vulnerability present in widely deployed meters in a large-scale coordinated attack. Such an attack could have immediate and costly consequences. However, fixing such a bug would also be enormously expensive: current systems require physical access to reliably update firmware and repair or replace physical components. In the few systems that do support over-the-wire firmware updates, compromised systems can silently refuse the updates [16].

Many have sought to prevent common failure modes present in monocultures by introducing artificial software diversity [9, 13, 1, 18, 4]. Unlike "real" software diversity that requires nodes to run independently developed software, artificial diversity alters the system software image such that the each instance or execution is unique. Canonical diversity techniques randomize the in-memory layout of code and data. Because addresses are unpredictable, customized attacks have to be crafted for each victim system–an effort intensive proposition.

Existing diversity techniques can not be directly applied to smart meters. This is because these techniques require advanced processor features such as advanced programming environments, memory management and protection rings. For example, protections based on safe C variants [9, 13] or the addition of hardware-based fault isolation is not practical in commercial embedded development environments. Moreover, instrumentation such as control-flow integrity [1], data flow integrity [4], and system callsite verification [18] require a tamperproof monitor to enforce the intended code and data paths. Firmware-diversity on the other hand, is minimally invasive and can

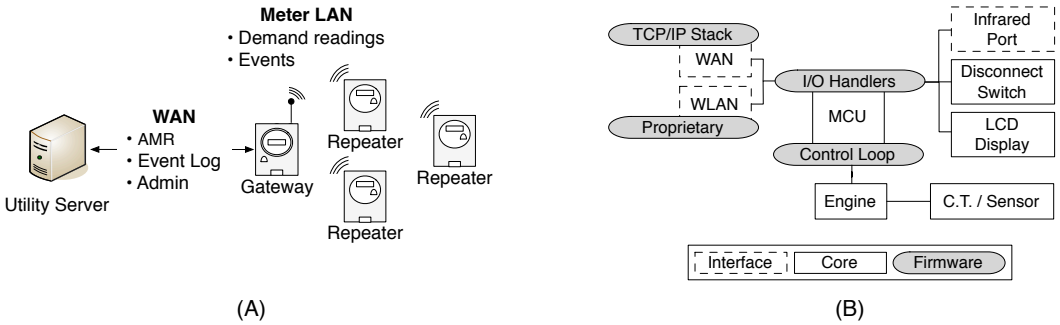

Figure 1: Smart meter operation. Figure (a) shows the two-tiered structure of meter-utility communications. Figure (b) shows a typical smart meter architecture.

be implemented through static binary rewriting.

In this paper, we consider how diversity techniques can limit large-scale attacks on the smart grid. We explore the capabilities of an adversary attempting to mount a large-scale attack. A novel redundant address encryption diversity technique suited to the computationally and resource limited smart meters is discussed, and its effectiveness is analytically evaluated. Preliminary results show that even when assuming a resource-rich attacker with knowledge of meter vulnerabilities and unfettered access to meter networks, large-scale compromises can be hampered significantly. We begin in the next section with a brief overview of the smart grid and meter architectures.

2 Smart Grids

This section details the features of commercial meters. It then reviews the impact of the smart meters underlying hardware on firmware diversity techniques.

2.1 Grid Architectures

Smart meters reduce operating costs for utilities. Primarily, they provide automated meter reading (AMR), in which network-connected meters report power usage remotely. Along with usage information, smart meters can also notify utilities of events such as power outages, power quality problems, and potential meter tampering. Smart meters can record time of day information along side usage to allow for time-sensitive pricing [15]. Besides timekeeping capabilities, this also requires onboard storage to record net and peak usage at intervals normally ranging from one to fifteen minutes. Finally, meters can perform a number of unscheduled actions such as firmware and configuration upgrades from the utility and disconnection of electrical service due to delinquency.

Smart meters communicate with utilities over a two-tier network shown in Figure 1.a. Most meters reside in the edge tier interconnected via a wireless mesh network. Less common deployments use power line communication to form a local area network of *repeater* meters. The purpose of the edge tier is to propagate information to the *gateway* meter. The gateway forwards data and signaling to and from the utility over a public network such as the Internet or cellular network. Gateways–also called collectors–may or may not function as meters and may proxy data for a few to over a thousand meters (depending on the vendor design and provider needs). Gateways are in almost all cases physically insecure. Both gateways and repeaters have infrared (optical) ports that are used for provisioning and maintenance by an onsite technician. Note that optical ports may be probed for exploits in the same manner as the wireless and WAN links.

2.2 Meter Design

A conceptual block diagram for a typical smart meter is shown in Figure 1.b. The meter's main control firmware runs on a (Microcontroller Unit) MCU that coordinates collection of usage data from the meter engine, storage of usage data and I/O. The controller communicates with the LAN and WAN cards via a modem-like serial AT command interface. Each of these runs an embedded network stack for the appropriate medium (Ethernet, IR, cellular, modem). The meter engine receives analog measurements from a pair of Current Transformer (CT) coils, and computes digital *pulse data* that is delivered to the meter's MCU at one second intervals. Energy samples are stored in flash memory and forwarded to the provider based on a configured schedule.

The main processor and communication cards exchange data over exposed buses contained within the meter enclosure. Each component's firmware is almost exclusively built upon commodity embedded operating systems and low cost processors similar to ARM processors but more minimal in features.

Demonstrative meters are shown in Table 1. The only feature-rich architecture found is in the gateway node.

Table 1: The operating environments for the different types of firmware in two commercial smart metering systems.

Firmware Type / Meter Vendor	Processor Type	MMU	Privileged Mode	NX Bit	RAM
Repeater Controller	Renesas M16C [7]	No	No	No	20KB
Wireless Mesh	Renesas H8S [6]	No	No	No	N/A
Embedded TCP/IP	Lantronix DSTni-EX 186 [11]	No	No	No	256KB
Gateway Controller	Intel i386EX [12]	Yes	Yes	No	8MB

The majority of meters rely on more cost-effective components that lack support of advanced processor features. For example, the lack of virtual memory combined with the small physical address space immediately rules out diversity techniques based on address space randomization [27, 14]. Similarly, the lack of a non-executable bit or W$\oplus$X bit allows code to be injected on the stack.

3 Software Diversity

The most common exploit used against commodity system is a buffer overflow. These attacks manipulate return addresses on the stack [2] to execute (often injected) code. Canaries, or "stack cookies" are random values placed between a function's local variables and the return address [25]. The value is checked for modification before the return address is followed to determine if a buffer overflow clobbered the return address. The assumption is that if the return address is modified, then the canary must have been modified. In an embedded architecture where the heap is not separated from the stack via segmentation, this is not necessarily the case. Canaries are particularly weak in the 8- and 16-bit architectures common in smart meters because they are practically guessable.

A similar attack, heap overflows, exploit function pointers on the heap to achieve the same ends. While it has been suggested that the fields of data structures on the heap be randomized [17], the small number of possible permutations does not stand up against continued probing. A more general solution, address space layout randomization (ASLR) [24], randomizes the locations of code in memory to make it difficult to predict the proper address to craft in the heap [10]. ASLR benefits from architectural support for large virtual memory spaces which create a large space in which to place code. ASLR can be circumvented either through brute force [27] or in a single attempt given memory disclosure of the global offset table used for dynamic linking [26]. While statically linked embedded firmware does not suffer from the latter problem, their limited physical address spaces, which are typically between 10 and 50KB in size, render them vulnerable to parallel brute force probing. Smart meters are equally poorly suited to support other layout obfuscation techniques such as stack frame padding [3].

A common defense against code injection is the non-exectable (NX) bit, which can be set on a program segment to prevent it from being executed as code. This renders injected code useless. NX bits require hardware level support from a memory management unit (MMU), which is not present in most MCUs used by smart meters.

4 Threat Model

We now describe the capabilities of an adversary attempting a large scale compromise of smart meters in order to derive a set of requirements for firmware diversity. The goal of the adversary is to compromise a large number n of remote meters. The adversary knows a vulnerability present in the meter firmware, but not the random secret used to diversify each meter. Let p denote the expected number of attempted exploits (*probes*) needed to compromise a single meter. We assume that the adversary can probe all n meters in parallel, be it through the WAN, meter mesh networks, or both, receiving notification if an exploit succeeds. Before we can model the time cost to full compromise, we must establish a rate of probing.

From our experience, we find that communication between utilities and meters is very infrequent, usually on the order of days. Similarly, communication with gateway meters occurs at most every fifteen minutes. Given these lax throughput requirements, we make the conservative assumption that meters can rate limit requests to r requests per second, per unique source address. Rate limiting must be done per source address to prevent a DoS attack in which a high volume of probes cuts off utility communication with the meter. While firewalls represent another defense against such attacks, we do not believe utilities can be expected to properly configure them in even small deployments (on the order of 1,000 collectors). Powerful adversaries, those with large numbers of machines with which to perform probing, may lessen the effect of per-connection rate limiting. Given these definitions, we can model the time cost T to compromise n remote meters as $T = \frac{pr}{n}$. In the following sections, we describe a method to obtain a satisfactory value for p, and evaluate it in light of a powerful attacker.

Regarding the power of each exploit attempt, we assume the attacker can arbitrarily overwrite control flow

information on the stack without being detected by stack protection mechanisms. Furthermore, we assume that memory disclosure may be used to obtain arbitrary memory contents, with the exception of the diversity secret. This is reasonable as unsafe `printf` usage can be curbed by padding the secret with `NULL` bytes, which cause `printf` to stop reading a string. We note that some meters do limit their attack surface by dropping unauthenticated requests without further processing. In practice however, we have found that some meters run multiple third party networking stacks, and support undocumented features such as an FTP server which caused frequent reboots in a collector unit that we subjected to fuzz testing.

5 Firmware Diversity

This section presents a method for firmware diversity capable of significantly slowing a large-scale compromise of smart meters. We propose a form of return address encryption to protect addresses on the stack that can be implemented via binary rewriting.

5.1 Address Encryption

Strong threat models, such as the one presented in this paper, assume an adversary can overwrite any data on the stack or heap without detection by any canary-like mechanism. A stronger protection method, as proposed by PointGuard ™ [5], is to encrypt these addresses before they are stored in the stack or heap, and decrypt them immediately before they are used by a branch instruction. If an exploit overwrites an address without knowing the decryption key, the decryption process will mangle the exploit address into a random value. The scheme works as follows. When a function is called, the return address A is not immediately written to the stack. Instead it is combined with a secret key K using exclusive or, and the resulting $A \oplus K$ is pushed to the stack. K is the only secret in memory that must be protected from memory disclosure attacks. Before the function returns, $A \oplus K$ is read into a register and recombined with K to recover the original address, which is then used for the branch back to the calling function. While exclusive or may not seem like an adequate encryption method, the adversary only receives notification if the attack was a success or failure, forcing a brute-force exploration of the entire key space.

There is one major shortcoming of using this method in isolation. If an adversary uses a vulnerability to overwrite $A \oplus K$ with some value A', the decryption process will yield $A' \oplus K$, a random address. In the original work on PointGuard, it is assumed that in a sparse virtual address space, jumping to such a random address will result in a segmentation fault or other type of program crash. This is considered a preferable fail-stop compared with a successful compromise. However, because smart meter downtime equates to lost revenue for utilities, such crashes or other unpredictable behaviors are unacceptable beyond a few meters. If a random guess at an address could be used to crash a meter, then an adversary could launch a large-scale "ping of death" attack against an entire smart meter installation with high efficiency. A method to validate a recovered address is needed.

5.2 Redundant Address Encryption

One way of validating decrypted addresses would be to keep a hash of the valid address along with the encrypted copy. Hashing however, is too heavyweight to execute at every function invocation. A more lightweight mechanism such as a checksum will be modifiable by an exploit, and thus will not detect tampering. Instead, we design a scheme that uses redundant encryptions of the same address with different keys. As a compromise between attack slowdown and meter performance, we use three encryptions total with keys K_1, K_2, and K_3. More or less redundancy may be used depending on the threat model and performance requirements. The resulting address kept on the stack is $A \oplus K_1 || A \oplus K_2 || A \oplus K_3$. Two additional machine words are allocated for the extra addresses. When an adversary overwrites this address using $A'_1 || A'_2 || A'_3$, the resulting decryption will be $A'_1 \oplus K_1 || A'_2 \oplus K_2 || A'_3 \oplus K_3$. Before the branch is taken, a check is performed that $A'_1 \oplus K_1 = A'_2 \oplus K_2 = A'_3 \oplus K_3$. If any one of the three decrypted address does not match another, then an attempted exploit has occurred, and the branch is not followed. Note that the adversary is forced to overwrite all three encrypted addresses because all three result in the same valid control flow. The probability of a false negative, three matching addresses that are not a valid control flow, is $1/2^{32}$ on a 16-bit architecture. The probability that a meter is successfully compromised is $1/2^{48}$, sufficient to severely limit the rate of an attempted large-scale compromise.

What is left is the question of how to react to an invalid decryption. Most commercial smart meters already support logging and some form of alarm condition reporting, but do not have a means of responding to detected intrusions without a human in the loop. We believe that the most reasonable defense is to immediately stop processing the request, which is at this point unauthenticated, and generate a new K_1, K_2 and K_3.

5.3 Binary Instrumentation

Due to the use of firmware from different third party vendors, it is impractical to implement redundant address encryption in each compiler. Instead, we aim to achieve it using binary instrumentation. We provide an example

Original function call:

```
push  A              ; Save address
jmp   B              ; Perform branch
```

Instrumented function call:

```
mov   D [key1_addr]  ; D = K_1
mov   C A            ; C = A
xor   C D            ; C = C XOR D
push  C              ; Save encrypted address
mov   D [key2_addr]  ; D = K_2
mov   C A            ;
xor   C D            ; Second redundant encryption
push  C              ;
mov   D [key3_addr]  ; D = K_3
mov   C A            ;
xor   C D            ; Third redundant encryption
push  C              ;
jmp   B              ; Perform branch
```

Figure 2: Function Call Instrumentation

Original return:

```
pop   A              ; Load return address
jmp   A              ; Perform branch
```

Instrumented return:

```
mov   D [key3_addr]  ; D = K_3
pop   A              ; Load third encrypted address
xor   A D            ; A = A XOR D
mov   D [key2_addr]  ; D = K_2
pop   B              ; Load second encrypted address
xor   B D            ; B = B XOR D
mov   D [key1_addr]  ; D = K_1
pop   C              ; Load first encrypted address
xor   C D            ; C = C XOR D
cmp   A B            ; Check A - B
jnz   fail_stop      ; Fail if A - B != 0
cmp   B C            ; Check B - C
jnz   fail_stop      ; Fail if B - C != 0
jmp   A              ; Return to calling function
```

Figure 3: Function return instrumentation

here of the instructions that must be matched and modified at each function call and return. For this example, let register `A` contain the instruction pointer and register `B` contain the address of the target function. The ith secret key is located at `keyi_addr`. While the prototypical instructions most resemble the familiar x86 assembly language, equivalents exist for each architecture found in Table 1. In reality, the task is slightly easier on most architectures that support explicit `call` and `ret` instructions.

Figure 2 shows the instrumentation for a function call. In order to instrument calls, the actions taken to save the return address before the branch must first be identified. Some architectures support a single `call` instruction to perform both the save and the branch. In this case, it suffices to match against the explicit `call`. The placement of three return addresses does not affect instructions working with function local variables, because the stack base pointer is normally placed above the return address. The instrumentation for safe function returns is shown in Figure 3. Matching returns requires finding an explicit `ret` instruction where supported, or finding `jmp` that uses a register who's last value was popped from the stack.

While we acknowledge that code bloat is a real problem in embedded environments, we believe that binary instrumentation is still practical. There are several obvious improvements including rolling the encryptions into loops, and implementing the address encryption at the beginning of each function, as opposed to each of the far more numerous call sites. We defer any further improvements to future work.

6 Evaluation

Using firmware diversity, a global slowdown in a large scale compromise means a local performance impact for each meter. We now evaluate this trade off for a triple redundant address encryption scheme.

6.1 Attack Slowdown

In a simulation of a zero-day smart meter attack leveraging a real exploit, it was shown that a smart meter worm could propagate via the meter WLAN to 15,000 homogeneous nodes in approximately 24 hours [8]. Using the threat model described in Section 4, we observe a substantial increase in the time for a large compromise given diversified meters, even when assuming a much stronger attacker. Triple redundant address encryption affords an expected number of probes to compromise of $p = 2^{48}$. Assuming an attacker that controls one thousand machines, each of which can execute one parallel probe per minute, this would require approximately ten years to compromise an $n = 15{,}000$ node deployment in which per-source requests are limited to $r = 3$ per minute. This is highly conservative in light of the far less stringent throughput requirements for current smart meters. This leads us to believe that triple redundant address encryption with rate limiting is a sufficient diversity technique to be deployed in current meters, which are expected to work without field maintenance for decades.

6.2 Performance Considerations

The overhead created by redundant encryption can be inferred largely from examining the assembly code in section 5.3. In each case, the overhead is less than 15 cycles for the individual instructions plus the cost of the memory accesses to the three keys. Assuming that these values are frequently in the MCU cache, they should cost only one or two additional cycles. While this number of total cycles may be unacceptable on general purpose systems which have CPU-bound workloads, smart meter firmware

is mainly I/O-bound. The limited throughput of the network related firmware has already been discussed, and data is received from the meter engine at a rate of once per second to be stored in low-cost flash memory. The twenty cycles consumed by address encryption and decryption is negligible by comparison.

7 Summary and Future Work

This preliminary work has begun to consider diversity as a means of stemming the inevitable attacks against low cost smart meters. Future efforts will apply this and identify new techniques for diversity in embedded devices with feature-poor processors. Such efforts will be informed by the new attacks discovered in the field and within penetration testing. In preventing adversaries from leveraging vulnerabilities found in one system against others, we hope to mitigate large-scale infrastructure attacks that could damage communities, regions and nations.

Acknowledgements: We would like to thank Seth Blumsack for his insights into electric utility operations. This material is based upon work supported by Lockheed Martin. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Lockheed Martin.

References

[1] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti. Control-Flow Integrity. In *CCS '05*.

[2] Aleph One. Smashing the Stack for Fun and Profit. *Phrack*, 7, November 1996.

[3] E. Bhatkar, D. C. Duvarney, and R. Sekar. Address Obfuscation: an Efficient Approach to Combat a Broad Range of Memory Error Exploits. In *Proc. of the 12th USENIX Security Symposium*, 2003.

[4] M. Castro. Securing Software by Enforcing Data-flow Integrity. In *OSDI '06*.

[5] C. Cowan, S. Beattie, J. Johansen, and P. Wagle. PointGuard™: Protecting Pointers from Buffer Overflow Vulnerabilities. In *Proc. of the 12th Usenix Security Symposium*, 2003.

[6] R. Electronics. H8/300 Programming Manual. `http://www.informatik.uni-kiel.de/fileadmin/arbeitsgruppen/realtime_embedded/mindstorms/h3ltp001d1.pdf`.

[7] R. Electronics. M16C Software Manual. `http://documentation.renesas.com/eng/products/mpumcu/rej09b0137_m16csm.pdf`.

[8] K. Fehrenbacher. Smart Meter Worm Could Spread Like A Virus. `http://earth2tech.com/2009/07/31/smart-meter-worm-could-spread-like-a-virus/`.

[9] J. S. Foster, M. Fähndrich, and A. Aiken. A Theory of Type Qualifiers. *SIGPLAN Not.*, 34(5):192–203, 1999.

[10] A. Francillon and C. Castelluccia. Code Injection Attacks on Harvard-Architecture Devices. In *CCS '08*, New York, NY, USA. ACM.

[11] gridconnect. `http://site.gridconnect.com/docs/EX_Chip/DSTni-EX_UG_a2.pdf`.

[12] Intel. Intel386™Microprocessor. `www.intel.com/design/intarch/datashts/27242007.pdf`.

[13] T. Jim, J. G. Morrisett, D. Grossman, M. W. Hicks, J. Cheney, and Y. Wang. Cyclone: A Safe Dialect of C. In *Proceedings of the USENIX Annual Technical Conference*, 2002.

[14] C. Kil, J. Jun, C. Bookholt, J. Xu, and P. Ning. Address Space Layout Permutation (ASLP): Towards Fine-Grained Randomization of Commodity Software. In *ACSAC '06*.

[15] C. S. King. The Economics of Real-Time and Time-of-Use Pricing For Residential Consumers. Technical report, American Energy Institute, 2001.

[16] M. LeMay, G. Gross, C. A. Gunter, and S. Garg. Unified Architecture for Large-Scale Attested Metering. In *HICSS '07*.

[17] Z. Lin, R. D. Riley, and D. Xu. Polymorphing Software by Randomizing Data Structure Layout. In *Proceedings of the 6th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2009.

[18] C. M. Linn, M. Rajagopalan, S. Baker, C. Collberg, S. K. Debray, and J. H. Hartman. Protecting against unexpected system calls. In *Proceedings of the 14th USENIX Security Symposium*, 2005.

[19] M. A. Lisovich, D. K. Mulligan, and S. B. Wicker. Inferring Personal Information from Demand-Response Systems. *IEEE Security & Privacy Magazine*, 8, 2010.

[20] Y. Liu, P. Ning, and M. K. Reiter. False Data Injection Attacks against State Estimation in Electric Power Grids. In *CCS '09*.

[21] P. McDaniel and S. McLaughlin. Security and Privacy Challenges in the Smart Grid. *IEEE Security & Privacy Magazine*, 7, 2009.

[22] S. McLaughlin, D. Podkuiko, and P. McDaniel. Energy Theft in the Advanced Metering Infrastructure. In *Proceedings of the 4th International Workshop on Critical Information Infrastructure Security*, Bonn, Germany, September 2009.

[23] R. Meritt. Stimulus: DoE readies $4.3 Billion for Smart Grid. *EE Times*, February 2009.

[24] PaX Team. PaX. `http://pax.grsecurity.net/`.

[25] G. Richarte. Four Different Tricks to Bypass StackShield and StackGuard Protection. *World Wide Web*, 1, 2002.

[26] G. F. Roglia, L. Martignoni, R. Paleari, and D. Bruschi. Surgically returning to randomized lib(c). In *ACSAC '09*.

[27] H. Shacham, E. jin Goh, N. Modadugu, B. Pfaff, and D. Boneh. On the effectiveness of address-space randomization. In *CCS '04*.

[28] K. Zetter. Security Pros Question Deployment of Smart Meters. *Threat Level: Privacy, Crime and Security Online*, March 2010.