CHAPTER 1

Cyber Security in the Mobile Age

The number of new security threats identified every month continues to rise. We have concluded that security has now become the third pillar of computing, joining energy-efficient performance and Internet connectivity in importance.

—Paul S. Otellini

This book is an in-depth technical introduction to an embedded system developed and manufactured by Intel Corporation. The embedded system is not an independent product; it is a native ingredient inside most of Intel's computer product portfolio, which includes servers, desktops, workstations, laptops, tablets, and smartphones. Although not well known to most end users, the embedded system plays a critical role in many consumer applications that people use every day. As such, its architecture, implementation, and security features are worth studying.

Depending on the end product in which the embedded engine resides, the engine is denominated differently:

- For the embedded system shipped with computing devices featuring Intel Core family microprocessors, it is called the *management engine*.
- For the embedded system shipped with computing devices featuring the Intel Atom system-on-chip (SoC), it is called the *security engine*. Note that not all Atom platforms use the security engine introduced in this book.

For the sake of convenience, this book refers to it as *the security and management engine*, *the embedded engine*, or simply *the engine*.

Three Pillars of Mobile Computing

In August 2010, Intel announced the acquisition of security giant McAfee. Paul S. Otellini, Intel's president and CEO at the time, emphasized that "security has become the third pillar of computing" when commenting on the investment. The other two pillars of computing are energy-efficient performance and Internet connectivity.

The three pillars summarize the core characteristics for computing, especially mobile computing. Intel's security and management engine is an embedded component that serves as the backbone that supports the three pillars for multiple forms of computers, including mobile endpoints, desktops, workstations, and servers. As its name indicates, the engine's main functionalities are security and management. In the meantime, power efficiency and connectivity are also addressed in its design.

Power Efficiency

Mobile devices distinguish themselves from stationary platforms in mobility and independence of AC (alternating current) power supply. The battery life is hence an important factor for evaluating the quality of a mobile product. Before the battery technology sees a major breakthrough, computer manufacturers have to strive to deliver hardware and software with low energy consumption.

A number of general strategies can be employed to save power:

- Decrease the processor's clock frequency, with the potential tradeoff of performance. For example, the security and management engine runs at a significantly lower speed than the platform's main processor. This is possible without degrading the user experiences, because the engine is not designed to be involved in performance-critical paths.
- Dim the display screen and shut down devices that are not being used or place them in sleep states. For example, after being idle for a configurable amount of time, like 30 seconds, the security and management engine may completely power off or run in a low-power state with very low clock frequency. Events that may wake up the engine to its full-power state include device interrupts and messages received from the host operating system.
- Simplify and adjust hardware and software logic. Redundant routines should be removed. For example, applying blinding to public key operations is meaningless, because there is no secret to be secured from side-channel attacks; whenever feasible, favor performance over memory consumptions for runtime programs. These are part of the design guidelines for the security and management engine.

Internet Connectivity

Needless to say, the majority of applications running on a mobile device rely on network connections to function. Looking into the architecture, there are two models of splitting the workload between the local device and the cloud:

- The main functionality of the cloud is storage, for contents such as movies, music, and personal files. The local device carries out most of computational tasks. This model requires stronger computing capability of the mobile devices, which may imply higher prices.
- Besides storage, the cloud also performs a certain amount of computations for the device. The device is responsible for only limited computations, and its main tasks are input and output. This model is advantageous in lowering the cost of the device. However, it requires high network bandwidth and powerful servers that are able to support a large number of devices simultaneously.

Security

Security is not standalone, but closely relevant to the other two pillars. Security is becoming vitally important for computers, thanks to the increasing connectivity. While enjoying all the benefits and conveniences the Internet has to offer, connected devices are also exposed to widespread attackers, viruses, and malware on the open network. The new challenges of securing mobile platforms are originated from three characteristics of mobile computing:

- *Always connected*: Smartphones and tablets may never be turned off. Attacks can be mounted at any time and take any amount of time.
- *Large data transmission*: Because of its convenience, mobile devices are used more often for operations that involve secure data transmission with servers, for example, web site logon, financial transaction, online purchase, and so forth. This makes attacks that require collecting a large amount of data more likely to succeed.
- *Privacy*: Mobile devices hold sensitive data that would not normally appear on stationary computers. The data includes but is not limited to phonebook and location information. A security objective for mobile devices is to protect users' personal information.

To mitigate these threats, security researchers have invented and deployed various countermeasures to safeguard computers and prevent leakage and abuse of assets. They include software-based solutions, like antivirus programs, firewalls, and so on, and hardware-based solutions, such as secure boot.

Now let's take a look at the relationship between security and power. Unfortunately, improvements in security and reduction in energy consumption are largely contradictory. A security measure, although an essential element, costs power to accomplish its work that is not functionally beneficial. However, an insecure system is not practically usable. Well-designed cryptography and security implementations can provide desired protection strengths with minimum power consumption. The following are some strategies that can be considered:

- Offload intensive mathematical operations to hardware engines that operate at lower frequency. Most cryptography algorithms are built on complex mathematics. The dedicated hardware should feature specific logic for underlying operations, so the calculation can be completed faster with lower power, compared to general-purpose processors.
- Utilize efficient algorithms and parameters; for example, when designing elliptic curve cryptography, select the curves carefully, and use the ones that require the fewest operations without degrading the security strength.
- Avoid overengineering. Choose algorithms and key sizes that meet, but don't overwhelmingly exceed, robustness requirements. For example, using a public key cryptosystem with security strength of 256 bits to protect a 128-bit symmetric key is a waste of computing power.
- Store keys and other secrets in secure, nonvolatile memory if possible and avoid repeated derivations for every power cycle.

BYOD

Bring Your Own Device, or BYOD, is a fast-growing emerging application thanks to the booming mobile computing development. An increasing number of companies now support BYOD programs and allow employees to use their personal mobile devices for work, such as sending and receiving corporate e-mails and accessing work data.

According to a survey[1] conducted by Intel, the following are the three top-ranked benefits voted by corporate IT (information technology) managers across different continents:

- Improve efficiency and worker productivity
- Increase opportunities for worker mobility
- Save costs by not having to purchase as many devices for employees to use

Alongside the gains are risks and challenges. Not surprisingly, security is the number-one rated barrier of deploying BYOD in most countries, especially for heavily regulated industries. With BYOD, it is increasingly common to see malware that targets the IT infrastructures of government agencies and industrial companies. The safety level of a corporate asset is equal to the strength of the weakest link that handles the asset. Because the employees' devices are handling confidential business data, they must apply the required security enforcements per the company's IT policies.

Here are a few security considerations when converting an employee's device for BYOD:

- *Secure boot*: The system integrity must be guaranteed. Rootkits and malware that infects the boot flow place the entire operating environment at risk. It is recommended that rooted mobile devices should not be permitted for BYOD. Refer to Chapter 6 for technical insights into Intel's Boot Guard technology.
- *Hard-drive encryption*: The whole drive, or at least the partition that stores business data, should be encrypted with a standard algorithm. The encryption key may be randomly generated at the first boot and sealed in a dedicated trusted device, such as a TPM[2] (*Trusted Platform Module*). The key may also be calculated from the user's credentials using a one-way function with a salt at each boot. Regardless of how the key is generated, it should be unique per device. Deriving the key solely from a password is not a good idea, because the employee may use the same password for multiple purposes.
- *Strong authentication*: The minimal length and complexity of the login password should be enforced. A password should be a combination of characters and cannot be a four-digit number. The device memory should not contain plaintext secrets before the device is unlocked by the employee. In addition, some business applications may warrant additional multifactor authentication at runtime.
- *Isolated execution*: Sensitive applications should execute in a secure mode that is logically separated from the nonsecure mode and other irrelevant applications. Intel's proprietary features, like TXT[3] (*Trusted Execution Technology*) and the upcoming SGX[4] (*Software Guard Extensions*) technology, have built infrastructures for isolated execution.
- *Employee privacy*: Depending on the organization's BYOD policy, the employee's personal data, such as photos, videos, e-mails, documents, web browse cache, and so on, may need to be secured from access or abuse by business applications. This can be achieved by the same isolation technique mentioned earlier.

- *Remote wipe capability*: Mobile device theft is on the rise, rapidly. *Consumer Reports* projects that about 3.1 million American consumers were victims of smartphone theft in 2013, more than double the 1.4 million in 2012. Once a BYOD device is reported stolen, even though the hard drive is encrypted, it is still essential, for defense in depth, to wipe the hard drive and prevent loss of business data and personal information. In April 2014, major industry players, including Apple, Google, Microsoft, Samsung, Nokia, AT&T, Sprint, and others, signed on to the "Smartphone Anti-Theft Voluntary Commitment"[5] that pledges to implement a "kill switch" feature by 2015 that can wipe the data of a lost phone and disallow the phone from being reactivated without an authorized user's consent.

While tightening up the security of employees' mobile equipment and getting ready for BYOD, an inevitable side effect is the increased power consumption and worsening battery life. To improve employee satisfaction, the strategies discussed in the previous section should be taken into account when defining BYOD policies.

Incident Case Study

What's happening in the area of cyber security? From credit card fraud to identity theft, from data breach to remote execution, cyber security is being covered increasingly by the media—not only technical journals but also popular newspapers and magazines—and is drawing a lot of public attention. The subject of cyber security is no longer an academic matter that concerns only researchers and computer manufacturers. In the era of mobile computing, cyber security is a problem that impacts everyone's life more realistically than ever.

eBay Data Breach

In a press release from May 21, 2014, the giant Internet auction house eBay said it would ask its 145 million customers to change their passwords, due to a cyber-attack that compromised a database containing encrypted passwords and other nonfinancial data.[6]

How did it happen? According to the press release, the cyber-attack occurred between February and March of 2014. It comprised a small number of employee login credentials, allowing unauthorized access to eBay's corporate network. The company later clarified that the passwords were not only "encrypted," but also protected by eBay's "sophisticated and proprietary hashing and salting technology," and there was no evidence that the stolen data could be used in fraudulent actives.

Despite the fact that the stolen passwords were protected (encrypted and hashed), there are still a series of implications of the incident:

- Users' private information—including name, postal and e-mail addresses, phone number, date of birth, and so forth—was stored in the clear and leaked.

- Depending on the encryption or hashing algorithms (which are not disclosed by eBay) that are used to protect passwords, dedicated attackers may be able to reverse-engineer the algorithms and retrieve clear passwords.
- Password reuse among multiple sites is a poor but extremely popular practice. Even if a victim changes her password for eBay.com, her cyber security is still in danger if she also uses the same password for other web sites, such as Amazon.com. Therefore, an eBay user must change passwords for all web sites for which the compromised password is used, to be safe.

Target Data Breach

On December 19, 2013, Target, the second largest retail store in the United States, reported a data breach that resulted in approximately 40 million credit and debit card numbers being stolen.[7] Victims were consumers who shopped at Target stores (excluding Target.com) between November 27 and December 15, 2013 and paid with payment cards. In January 10, 2014, the company further announced that, in addition to the 40 million stolen cards, personal information, including names, phone numbers, and postal and e-mail addresses of approximately 70 million customers, was also compromised due to the incident. In other words, nearly one-third of the total population of the United States was impacted.

Following one of the most massive breaches in US history, in February 2014 Target reported that its net earnings for the holiday season had plunged 46 percent year-to-year. On March 5, the company's chief information security officer resigned from the job. Two months later, Target's chairman, president, and CEO Gregg Steinhafel also stepped down, after as many as 35 years of service at the company. The press release described that Steinhafel "held himself personally accountable" for the breach.

The company explained in January 2014 that the breach was due to login credentials being stolen from one of its vendors and then used to access Target's internal network. The attacker might have exploited vulnerabilities in the management software deployed in the internal network to access the secret data. Target did not disclose the name of the vendor or the management software.

From the brief description, one may reasonably deduce what happened: the attacker logged into Target's network using the stolen credentials. He then installed malware on the flawed management software to exploit the vulnerability. The malware scanned in the host memory for payment card numbers and then secretly uploaded to a remote server established by the attacker that harvested them. Furthermore, the fact that online purchases at Target.com were not affected suggested that the malware might have infected point-of-sale (POS) machines. A research conducted by Intel's McAfee Labs following the incident had identified a number of malware that aims at POS endpoints and transaction verification systems.[8]

The breach unfolded several problems with Target's information security management:

- Vendors' access to Target's network was not protected with a sufficiently strong authentication method. A credential that can be stolen and used without the victim's knowledge is likely a username and password. This old and simple way of authentication is very vulnerable and too weak to fortify valuable assets.
- The vendor's account was allowed to perform privileged operations after accessing Target's internal network, and the operations were not closely monitored and examined. The principle of least privilege should always be exercised as a best practice of information security, not only in computer product designs but also in enterprises' IT management.
- The third-party management software suffered critical security flaws. Either the software vendor did not know about the vulnerability until the breach took place or Target did not apply patches that fixed the vulnerability in a timely manner.

OpenSSL Heartbleed

The Request for Comments 6520 "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension,"[9] published by the Internet Engineering Task Force (IETF) in February 2012, introduces and standardizes the *heartbeat extension* for the TLS/DTLS protocol. In a nutshell, it is a simple two-way protocol between a client and a server that have already established a secure TLS/DTLS session. One party sends a heartbeat request message with an arbitrary payload to its peer, who in turn sends back a heartbeat response message that echoes the payload within a certain amount of time. This extension is mainly used for checking the liveliness of the peer.

The core of the mobile computing is interconnectivity—connections between a client (laptop, smartphone, tablet, and so forth) and a server, between two servers, or between two clients. There exist various protocols that offer secure links between two entities, for example, the SIGMA (*SIGn and message authentication*) protocol introduced in Chapter 5 of this book. However, TLS/DTLS is used in the majority of secure connections over the Internet today. It provides not only one-way or mutual authentication but also encryption and integrity for messages. Most implementations of TLS/DTLS take advantage of the open-source OpenSSL cryptography library.

Heartbleed is a severe security bug in OpenSSL.[10] The vulnerability was first reported by Neel Mehta of Google's security team on April 1, 2014. The Finnish cyber security company, Codenomicon, found the same issue independently at almost the same time and named it Heartbleed. The bug was fixed promptly in an OpenSSL release on April 7.

A heartbeat request message consists of four elements:

1. Message type (1 byte)
2. *payload_length* in bytes (2 bytes)
3. Payload (length is determined by *payload_length*)
4. Padding (at least 16 bytes)

The total size of a heartbeat request message is transmitted to the receiver in variable *TLSPlaintext.length* or *DTLSPlaintext.length*, whose value must not exceed 16384 according to the protocol. Notice that the 16-bit integer *payload_length* can denote up to 65535. The bug in the vulnerable OpenSSL releases lies in the receiver side of the heartbeat implementation. The code misses bounds checking and fails to make sure that the *payload_length* is such that the total size of the four fields is not greater than the actual message size indicated by *TLSPlaintext.length* or *DTLSPlaintext.length*. The flawed implementation outputs a response heartbeat message with memory buffer of size calculated based on *payload_length*, instead of *TLSPlaintext.length* or *DTLSPlaintext.length*.

To exploit the vulnerability, a malicious TLS/DTLS client assigns a small number to *TLSPlaintext.length* or *DTLSPlaintext.length*, manipulates *payload_length* to its allowed maximum, 65535, and sends the falsified heartbeat request to a vulnerable server. On the server side, nearly 64KB of the memory beyond the payload is transmitted back to the malicious client in the heartbeat response. Although the attacker cannot choose the memory region at which he can peek, the leaked memory likely contains information used for the current TLS/DTLS session, including the server's secrets. And the attacker can iterate heartbeat requests repeatedly to gather more memory content from the server. Similar attacks can be launched from a rogue server to attack flawed clients.

Figure 1-1 shows the attack scenario. The attacker's client first establishes a TLS/DTLS connection with the target server. The client then sends a manipulated heartbeat request to the server. The message is only 28 bytes in size, but it specifies, on purpose, the payload length as 65535—the maximum value that can be represented by a 16-bit integer—although the actual payload is only 9 bytes long: `{ed 15 ed 7c 05 9f 7b 99 62}`. In the OpenSSL implementation, the size of the padding field is fixed at the minimum allowed by the standard, 16 bytes.

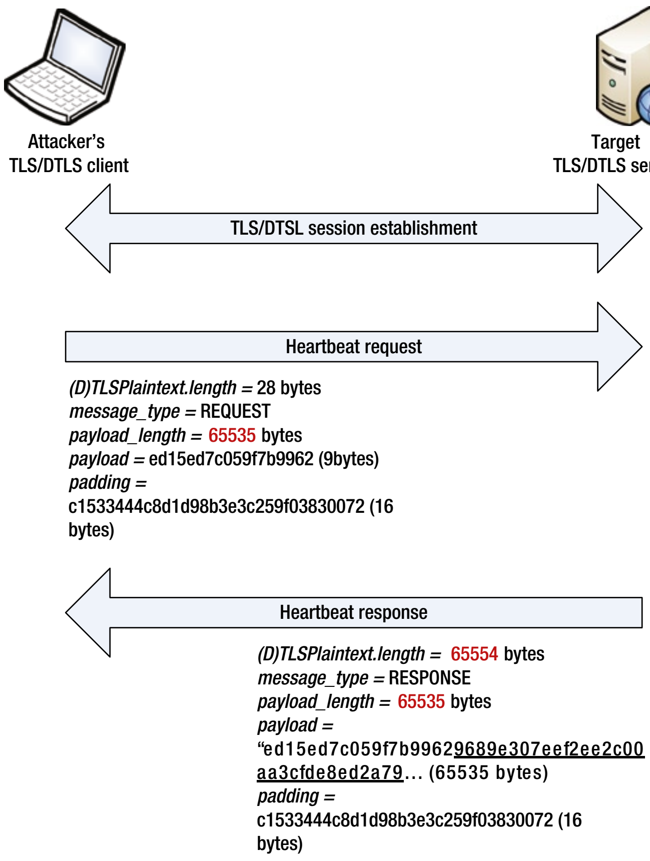
Attacker's
TLS/DTLS client
Target
TLS/DTLS server
TLS/DTSL session establishment
Heartbeat request
(D)TLSPlaintext.length = 28 bytes
message_type = REQUEST
payload_length = 65535 bytes
payload = ed15ed7c059f7b9962 (9bytes)
padding = c1533444c8d1d98b3e3c259f03830072 (16 bytes)
Heartbeat response
(D)TLSPlaintext.length = 65554 bytes
message_type = RESPONSE
payload_length = 65535 bytes
payload = "ed15ed7c059f7b99629689e307eef2ee2c00aa3cfde8ed2a79... (65535 bytes)
padding = c1533444c8d1d98b3e3c259f03830072 (16 bytes)

Figure 1-1. *Heartbleed attack*

The server with a buggy OpenSSL library calculates the total size of the heartbeat response buffer by adding the sizes of the message type (1 byte), *payload_length* field (2 bytes), payload (*payload_length* bytes), and padding (16 bytes), which works out to be 1+2+65535+16=65554 bytes in this case. Due to the missing bounds check, the server fails to realize that the size of its response has exceeded the maximum, 16384 bytes, defined by the specification. The size of the response also exceeds the size, 28 bytes, of the received heartbeat request. That is, as many as 65535-9=65526 bytes of the server's memory (an illustrative example is underlined in the figure: {`96 89 e3 07 ee f2 ee 2c 00 aa 3c fd e8 ed 2a 79 ...`}) following the payload is sent to the client in the heartbeat response. The leaked memory could contain the server's private key.

The bug had existed in OpenSSL for over two years before it was discovered. The two most popular open-source web servers, Apache and nginx, both leverage OpenSSL and are hence vulnerable. Among all active Internet web sites in the world, two out of three use Apache or nginx, as reported by Netcraft's April 2014 Web Server Survey.[11] Affected sites include popular ones such as Yahoo! and Flickr. Other than web servers, OpenSSL is the dominant library embedded in many other types of networked computer products as well, such as secure teleconferencing devices. Intel's AMT[12] (*advanced management technology*) software development kit is also affected. Famous cryptographer Bruce Schneier described the incident as "catastrophic."

Unfortunately, fixing the bug on the servers is just the beginning of the firefighting. The certificates of impacted servers must be revoked by the issuing certification authorities, and new certificates must be issued for servers' new key pairs. In the worst case, if the server's private key had been stolen before the fix was applied and the attacker was also able to obtain TLS/DTLS session caches between the server and its (potentially a large number of) clients, then secrets transmitted in those sessions were also compromised. Typically, the secrets transported over TLS/DTLS are end users' passwords, financial transactions, credit card numbers, e-mails, and other confidential information. What's worse, there is no trace of whether Heartbleed exploitations had happened and when. Therefore, it is almost impossible to accurately assess the total loss caused by the bug due to its retroactive nature.

When it rains it pours. On June 5, 2014, OpenSSL released another security advisory for six recently reported and fixed flaws.[13] These bugs were more difficult to exploit than Heartbleed, but still drew significant media attention in the wave of Heartbleed.

Key Takeaways

What can we learn from the repeated cyber security crisis? How does a company fight against cyber-attacks that make it the headlines? How to protect users' safety on the Internet? Following is a postmortem on the recent incidents.

Strong Authentication

Organizations, such as law enforcement agencies, offline and online retailers, financial institutions, medical facilities, and so on, that possess and process high-value assets should consider implementing strong authentication for access control. A strong authentication mechanism would require multiple factors of credentials for logging in. The second credential factor is usually a physical object—for example, a token—that belongs to a legitimate user.

Today, multifactor authentication is no longer an expensive investment, thanks to the emergence of innovative technologies. For many applications, the potential monetary loss due to identity theft far surpasses the cost of deploying multifactor authentication. Chapter 10 discusses strong authentication in detail and Intel's unique and cost-effective solution to the problem—IPT[14] (*Identity Protection Technology*).

Network Management

Organizations should closely monitor all network activities and flag suspicious operations. Advanced firewall devices and antivirus programs should be employed to detect malware and respond correspondingly. Intel's AMT, a core member of the vPro technology, provides a hardware-based out-of-band platform management solution that reduces cost and simplifies network administrators' work.

Boot Integrity

A platform that has been infected by virus, malware, or rootkits is running in a state that is different from its trusted and known-good state. Secure boot mechanisms, available on most new computers, examine the integrity of the platform's firmware and software components during power-on. They are designed to detect changes in platform state and identify malicious programs on the system.

In addition, secure boot can collaborate with other security measures to store secrets inside hardware, so that the confidential data is available for use only if the platform is running in a trusted state.

Hardware-Based Protection

Sophisticated viruses are capable of scanning a system's memory for signatures of interesting data, such as transactions and payment card numbers. For software-based protections, sensitive data has to appear in the system's memory in the clear at some point to be consumed by software programs. The duration of the exposure may be very short but still enough for malware to do its work. Even though the data is properly protected during transmission and at rest, attackers only need to circumvent the weakest point.

The ultimate solution is to depend on hardware for security, in which case the secrets are never exposed in the system's memory in the clear. Successful attacks against hardware launched from a remote location, if not impossible, would require extremely advanced skills to find and exploit critical hardware vulnerabilities.

State-of-the-art computers are equipped with necessary devices and hardware-based countermeasures to safeguard users' confidentiality, at rest and at runtime. For example, the TPM serves as the hardware root of trust (see Chapter 7 for more information) for a platform; Intel's SGX technology allows software programs to create and run in dedicated enclaves that are inaccessible by other components, including ring 0 drivers.

Open-Source Software Best Practice

Besides open-source operating systems such as Linux, open-source implementations of standardized protocols and functionalities have become a mainstream. Open-source software is gaining widespread popularity on endpoint devices and clouds because of many advantages it fosters: low cost, maturity, allowing faster development cycle and reduced maintenance effort, and so on. Developers simply port the functional modules they need and integrate with their products, instead of writing from scratch. They usually do not have to dig into the internal details of how the libraries structure and function. All they need is to understand the API (*application programming interface*) and invoke it.

This practice is good and bad. Although it saves engineering resources, on the other hand, it also poses risks because engineers are blind to the code that they are responsible for. One of the major disadvantages of free open-source software is that the volunteering authors provide the source code as-is and are not liable for consequences of bugs in the code, therefore the users must exercise caution during integration.

Open-source software, especially those that have been used for a long period of time by a large number of commercial products, normally enjoys high quality and performance with regard to functionality. However, the security side is a completely different story. For software development, writing working code is a relatively easier job compared to security auditing that requires dedicated resources with specialized expertise for code review and penetration testing, which, due to funding shortage, is often inadequate for open-source software.

Many adopters do not exercise comprehensive security validation for open-source modules of the products like they do for their owned components. This is usually due to lacking an in-depth understanding of the open-source modules, which renders it difficult or impossible to come up with effective test cases that are likely to identify critical vulnerabilities. Another excuse for deprioritizing security validation on open source is the presumption, and de facto an illusion, that open-source software "must be" mature because it is open and can be read and reviewed by anyone, plus it has been deployed by countless other products for so many years. In reality, the openness does not imply secure code. The security validation gap of using open-source software should be filled by individual product owners.

Eventually, the amount of resources that should be spent on comprehending and validating open-source code is a judgment call about opportunity cost. If vulnerabilities are discovered in released products, will the expense of fixing the issue and deploying the patch be higher than the investment on validation? Notice that there is an intangible price of brand name damages that must be taken into consideration as well.

In the security and management engine's firmware, only a small fraction originates from open-source domain, and it is only used in modules that do not assume security responsibilities. For example, the TLS implementation in the AMT firmware application is not ported from OpenSSL and hence not affected by OpenSSL's vulnerabilities such as the Heartbleed. The validation of the engine does not discriminate between open source and closed source. Thorough testing is performed against open-source software used by the engine.

As a good general guideline, the technical white paper "Reducing Security Risks from Open-Source Software"[15] proposes five steps that organizations should go through to take advantage of open source and lower the associative risks:

1. Identify and analyze all usages of open source.
2. Assess open source for vulnerabilities ad resolve issues.
3. Develop open-source usage policies.
4. Develop a patch management process.
5. Create a compliance gate.

Third-Party Software Best Practice

Before purchasing commercial software from for-profit vendors or outsourcing software development to external parties, buyers should ask the following:

- What is the security development process exercised by the vendor?
- What types of security auditing are preformed? Is it done by the vendor itself or external consultants?
- What is the vulnerability tracking and response process?

Security validation is a pivotal stage in software development. A vendor with a good quality control system should apply proven techniques, such as static code analysis, penetration testing, and so forth, to their product development life cycle.

Even though the third-party software has been tested for security by its vendor, in many cases it is still worth it for the adopter to conduct independent code review and end-to-end validations, either in-house or by hiring specialized security auditing firms. This is necessary especially for modules that process sensitive data.

■ **Note** Consider performing comprehensive security validation and auditing for open-source and third-party software.

Security Development Lifecycle

The Security Development Lifecycle, or SDL, is a process consisting of activities and milestones that attempt to find and fix security problems during the development of software, firmware, or hardware. The SDL is extensively exercised by technology companies, for example, Microsoft and Intel. Different companies decide their specific procedures and requirements for SDL, but they all aim at the same goal: to produce high-quality products with regard to security and to reduce the cost of handling aftermaths for vulnerabilities found after release.

Intel is committed to securing its products and customers' privacy, secrets, and assets. To build a solid third pillar for computing, a sophisticated SDL procedure of five stages is implemented at Intel to make security and privacy an integral part of product definition, design, development, and validation:

- *Assessment*: Determine what SDL activities are applicable and will be performed.
- *Architecture review*: Set security objectives, list a threat analysis, and design corresponding mitigations.
- *Design review*: Map security objectives to low-level design artifacts. Make sure designs meet security requirements.
- *Development review*: Conduct a comprehensive code review to eliminate security vulnerabilities, such as buffer overflow.
- *Deployment review*: Perform security-focus validation and penetration testing and assure that the product is ready for release, from both the privacy and security perspectives.

The SDL process applies to hardware, firmware, and software, with small differences in different stages.

Intel takes users' privacy seriously. The privacy aspect is called out in the SDL process and evaluated separately, in parallel with the technical aspect of security, throughout all the five phases. Figure 1-2 shows the SDL phases and components. A product may ship only after the deployment privacy and security review has been accomplished and approved.

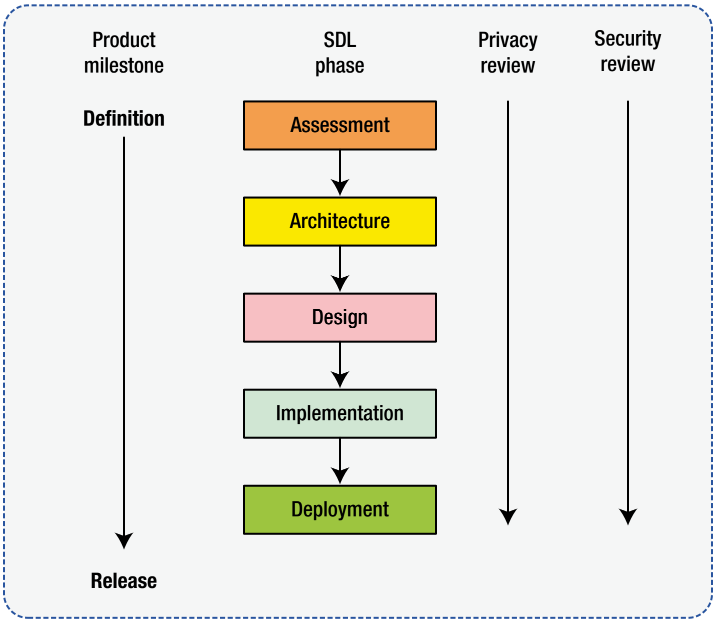

Figure 1-2. *SDL phases and components*

Assessment

The SDL assessment happens as part of the definition stage of a new product. The privacy assessment asks whether the product will collect users' personal information, and if so, what kinds of information, what it will be used for, and what techniques are employed to protect it from leakage and misuse. Intel has invented advanced technologies to safeguard users' fundamental right to privacy. Chapter 5 of this book is dedicated to privacy protection and Intel's EPID (*enhanced privacy identification*) scheme. The discussion in this section will focus on the security aspect of SDL.

Based on the nature and properties of the product, the assessment review concludes the set of SDL activities that must be conducted during the remainder of the product development life cycle. Generally speaking, a security feature—such as a TPM device or a cryptography engine—is subject to a complete SDL review, including architecture, design, implementation, and deployment. On the other hand, only select SDL stages may be required for those functions that are not sensitive to security per se, for example, Intel's Quiet System Technology (QST). Normally, architecture and design reviews may be skipped if the risk of waiving is deemed low; however, implementation and deployment reviews are almost always planned for all features.

Architecture

In this phase, the architecture owners of the product put together an intensive architecture description that presents the following points:

- *Security architecture*: The architecture includes components of the products, functionalities of each component, internal and external interfaces, dependencies, flow diagrams, and so on. A product's architecture is driven by its assets and functional requirements.
- *Assets*: Assets are valuable data that must be protected by the product, for confidentiality, integrity, and/or anti-replay. For example, the endorsement private key is a critical asset for a TPM and may not be exposed outside of the TPM. Notice that an asset is not necessarily the product's native value; it can also be users' data, such as passwords and credit card numbers. The security and management engine processes various types of user secrets and it is responsible for handling them properly per defined objectives.
- *Security objectives*: Security objectives are the goals that the product intends to meet for protection. For example, guarding the endorsement private key for confidentiality and integrity is a security objective for a TPM device; whereas thwarting denial of service (DoS) when an attacker is physically present is a not a security objective for the security and management engine.
- *Threat analysis*: Based on the in-scope security objectives, a list of possible attacker threats to compromise the objectives and assets are documented and analyzed. For example, in order to steal TPM's endorsement private key, an attacker may utilize side-channel attacks by accurately measuring power and time consumptions during a large number of the TPM's endorsement signing operations.
- *Mitigations against threats*: The mitigation plans detail how the architecture is designed to deter threats, protect assets, and achieve security objectives. In most cases, effective mitigations are realized through well-known and proven cryptography and security approaches. Note that security through obscurity is not a meaningful mitigation approach.

Figure 1-3 illustrates the components of the architecture review and relationships among them.

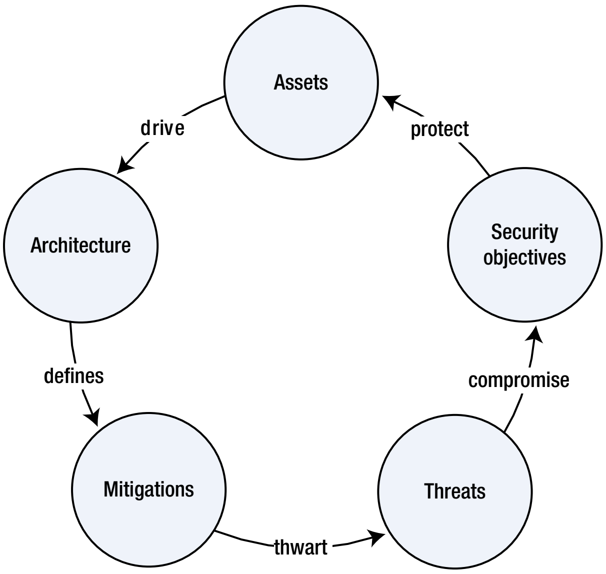

Figure 1-3. *Components of the architecture review and their relationships*

The architecture with aforementioned content is peer-reviewed and challenged by a group of architects and engineers with extensive experience and strong expertise in security. It is possible that a proposed new product be killed because one or more security objectives that are considered mandatory cannot be satisfied by a feasible and reasonable architecture. If and once the security architecture is approved, the SDL review process will move on to the design stage.

Design

During the design phase, high-level requirements are converted to prototypes. The design work for a software or firmware product contains a wide range of aspects. From the security perspective, in general, the most interesting ones are internal flows and external interfaces:

- *Internal flows*: A few security best practices should be followed in the flow design. For example: involve as few resources as possible; minimize dependency on shared objects and other modules; apply caution when consuming shared objects to prevent racing and deadlock conditions; avoid using recurrence on embedded systems.
- *External interfaces*: API must be defined with security in mind. For example: simplify parameters; do not trust the caller; always assume the minimum set of privileges that are needed to complete the tasks; verify the validity of input parameters before use; handle DoS attacks properly, if required.

Besides generic design principles, every product has its unique set of security objectives and requirements derived from the architecture review, which must be reflected in the design. The mitigations against threats and the protection mechanisms for assets are materialized in the design phase as well.

The design of cryptography should follow latest applicable government and industry standards. For example, encrypting data with AES[16] (*advanced encryption standard*); applying *blinding* to private key operations, if mitigation against timing attacks is an objective. Proprietary algorithms should be introduced only if absolutely necessary. Notice that use of nonstandard cryptography may pose difficulty in achieving security certifications such as the FIPS (*federal information processing standard*) 140-2 standard.[17]

Implementation

Engineers who implement the product in hardware or software languages should be knowledgeable about security coding practices. Members of the development team that is responsible for the security and management engine are required to complete advanced security coding classes and a training session on the security properties of the embedded engine, prior to working on the implementation.

Here are a few sample guidelines for software and firmware development:

- Use secure memory and string functions (for example, `memcpy_s()` instead of `memcpy()`) where applicable. Note that this recommendation does not apply to certain performance critical flows, such as paging.
- Comparison of two buffers should be independent of time to mitigate timing attacks. That is, `memcmp()` should process every byte instead of returning nonzero upon the first unmatched byte among the two buffers.
- Beware of buffer overflows.
- Make sure a pointer is valid before dereferencing it.
- Beware of dangling pointers.
- Beware that `sizeof(struct)` may result in a greater value than the total sizes of the structure's individual components, due to alignments.
- Set memory that contains secrets to zero immediately after use.
- Beware of integer overflows and underflows, especially in multiplication, subtraction, and addition operations.
- Remember bounds checks where applicable.
- Do not trust the caller's input if it is not in the trust boundary. Perform input parameter validation.

- When comparing the contents of two buffers, first compare their sizes. Call `memcmp()` only if their sizes are equal.
- Protect resources that are shared by multiple processes with mechanisms such as semaphore and mutex.

In addition, the development team that owns the security and management engine also observes a list of coding BKMs (*best known methods*) that are specific for the embedded engine. These BKMs are an executive summary of representative firmware bugs previously seen on the engine. It is crucial to learn from mistakes.

In practice, production code that is developed from scratch by an engineer with a secure coding mindset may be less of a problem. The more worrisome code in a product is usually those taken from nonproduction code. For example, proof-of-concept (POC) code created for the purpose of functional demonstration is often written with plenty of shortcuts and workarounds, but without security practice or performance considerations in mind. It is a bad practice to reuse the POC code in the final product, if and when the POC hatches to production, because in most cases, it is more difficult and resource consuming to repair poor code than to rewrite. Another source of possibly low-quality code similar to POC code is test code.

Following the completion of coding, the implementation review kicks off. The review is a three-fold effort: *static analysis*, *dynamic analysis*, and *manual inspection*. They may occur in tandem or in parallel.

The static analysis analyzes a software or firmware program by scanning the source code for problems without actually executing the program. A number of commercial static analysis tools are available for use for large-scale projects. If the checkers are properly configured for a project, then static analysis is often able to catch common coding errors, such as buffer overflows and memory leaks, and help improve software quality dramatically. However, despite its convenience, static analysis is not perfect. Particularly for embedded systems, because the tools in most cases do not fully comprehend the specific environments and hardware interfaces, a relatively large number of false positives may be reported. Notice that engineering resources must be allocated to review every reported issue, including those false positives. For the same reason, static analysis tools may fail to identify certain types of coding bugs.

In contrast to static analysis, dynamic analysis executes a software or firmware program on the real or a virtual environment. The analysis tests the system with a sufficiently large number of input vectors and attempts to exercise all logical paths of the product. The behavior of the system under test is observed and examined. Security coding bugs, such as a null pointer dereference, can be revealed when the system crashes or malfunctions. Such issues can be extremely hard to find without actually running the program.

The manual inspection is a source code review performed by fellow engineers that are familiar with the module, but did not write the code. The purpose is to make sure that the implementation correctly realizes the product's specific security design and architecture requirements. For example, an invocation of encryption for a secret is according to the specified algorithm, mode, and key size. Apparently, these kinds of issues cannot be found by the automatic tools. In addition, the review also checks that the generic coding guidelines are followed and tries to capture flaws in the code that were missed by the static and dynamic analysis.

As depicted in Figure 1-4, the implementation review is an iterative process. After functional or security bugs are fixed or other changes (like adding a small add-on feature) are made, the updated implementation must go through the three steps again. To conserve engineering resources, the manual code review may cover only the changed portions of the code. The two automatic analysis should be executed regularly, such as on a weekly basis, until the final product is released.

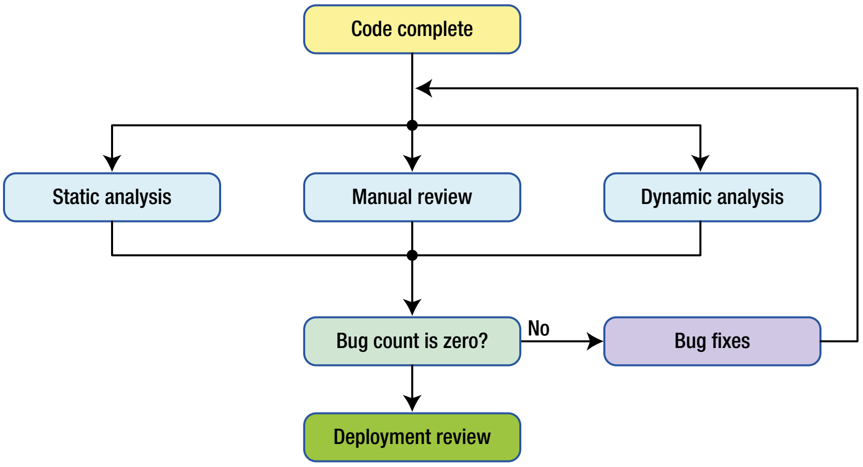

Figure 1-4. *Iterations of the implementation review. In this figure, static analysis, dynamic analysis, and manual review are performed in parallel*

Deployment

The deployment review is the last checkpoint before shipment. Sophisticated validations are performed against the product in this stage. The materials to help validation engineers create a test plan include output of the previous stages, such as security objectives of the architecture phase and the interface definition of the design phase. Comprehensive test cases aiming at validating the product's security behaviors are exercised.

Interface Testing

The first test object is the product's interface. Figure 1-5 is a graphical illustration of the interface test case design. Note that the output validation takes security objectives as input. A bug will be recorded when the behavior of the system under test violates one or more requirements.

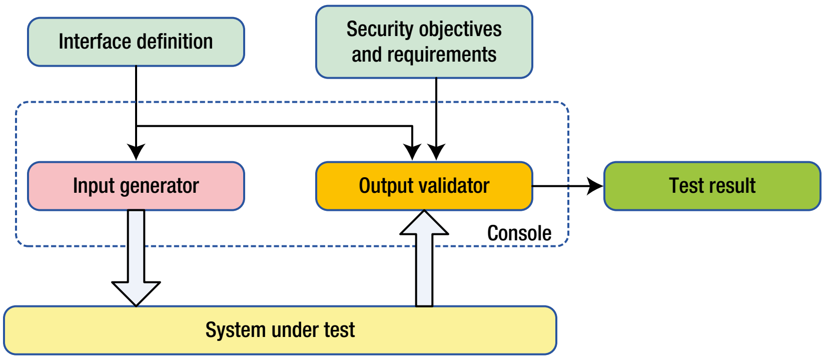

Figure 1-5. *Interface test design*

There are two categories of interface tests:

- *Positive test*: A positive test first invokes the product's interface with valid input vectors as specified in the design documents, and then verifies that the output from the system under test is correct per the security objectives and requirements. In most cases, there exist an infinite number of valid input value combinations. The test console may randomly generate valid inputs, but common cases (most frequently used values) and corner cases (extreme values) should be covered.
- *Negative test*: A negative test manipulates the input and invokes the product's interface with invalid input. The product is expected to handle the unexpected input properly and return an appropriate error code. It requires in-depth knowledge of the product in order to create good negative test cases that are able to expose security vulnerabilities.

To emphasize how pivotal the negative tests are, take the Heartbleed for example. Using the *Request for Comments 6520* as the requirement document, a simple negative test that acts like a malicious client that sends a heartbeat request with an excessive *payload_length* to the server under test would have caught the issue, because instead of notifying the client of an error (expected behavior), the flawed server would happily respond with its internal memory content of the illegitimate size requested by the client.

In addition to the scenario of "should bailout but does not," another common failure of negative tests is system crash, which can be a result of a variety of possibilities, for example, improper handling of invalid input parameters, buffer overflow, and memory leaks.

Fuzz testing, a kind of the negative testing, has become very popular in the recent years. The fuzz testing is an automated or semiautomated technique that provides a large set of invalid inputs to the product under test. The inputs are randomly generated based on predefined data models that are fine-tuned for the specific interface that will be tested. By looking for abnormal responses, such as crashing, security vulnerabilities such as buffer overflow can be uncovered.

Penetration Testing

The second type of tests intends to verify that the implementation is in accordance with the threat mitigation plan. A test of this type emulates an attack that is in the threat analysis of the architecture phase, observes the response of the product under test, and makes sure that it matches the behavior required by the mitigation plan. This type of testing is known as *penetration testing*, or *pentest* for short.

For example, the security and management engine reserves an exclusive region of the system memory for the purpose of paging. Any entity other than the engine changing the content of the region is deemed a severe security violation. Such an attack is considered and documented in the threat analysis, and the corresponding mitigation required is to trigger an instant power down of the platform as soon as the embedded engine detects the alteration.

A basic test for this case would flip a random bit in the reserved region of the host memory using a special tester and see whether the system indeed shuts down immediately as expected. Passing this basic test proves the correctness of the implementation at a certain confidence level. However, a more advanced test would understand the integrity check mechanism used for paging and replace the memory content with a calculated pattern that may have a higher chance of cheating the embedded system, and hence bypassing the protection. Obviously, design of such smart tests requires the knowledge of internal technical information of the product. This is called *white box* testing.

Before rolling out the product, a survivability plan should be drafted and archived. The survivability plan specifies roles, responsibilities, and applicable action items upon security vulnerabilities are found in the field.

CVSS

Even after going through stringent review and testing, vulnerabilities reported—either by internal teams or external sources—after the product is released are not uncommon. It is important to fairly evaluate the severity of escaped defects in order to take the right actions accordingly. For rating vulnerability, an industry standard used by the National Institute of Standards and Technology's National Vulnerability Database (NVD) is the CVSS[18] (*Common Vulnerability Scoring System*).

The CVSS defines three groups of metrics to describe vulnerability. They are base, temporal, and environmental, respectively:

- *Base group*: Represents fundamental characteristics of vulnerability. Such characteristics do not change over time or environment.
- *Temporal group*: Includes characteristics that change over time, but not environments.
- *Environmental group*: Covers characteristics that are relevant and unique to a particular environment.

Each group consists of several factors to be rated by the analysis. Figure 1-6 lists the factors under each group. The CVSS formula calculates a base score, a temporal score, and an environment score, respectively, from applicable factors. The calculation yields a score ranging from 0 to 10 inclusive, where a higher number indicates worse severity. According to NVD's standard, vulnerabilities are labeled "low" severity if they have a base score of 0.0 to 3.9, "medium" for a base score of 4.0 to 6.9, and "high" for 7.0 to 10.0.

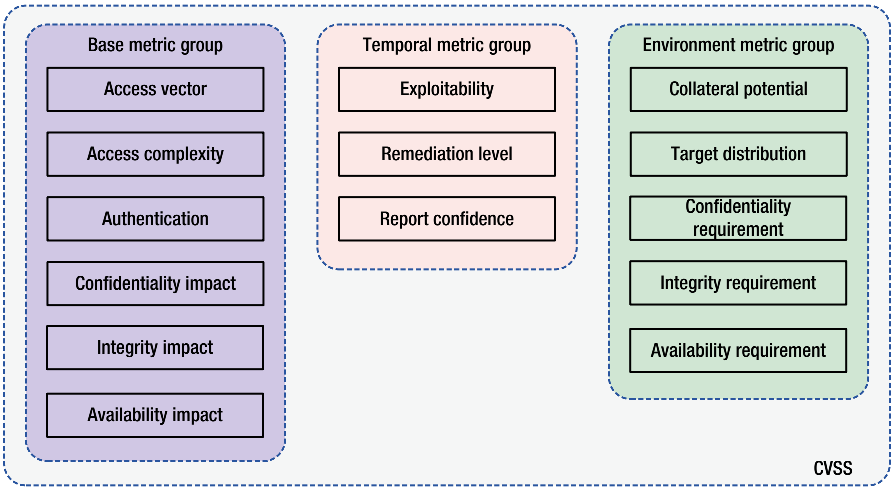

Figure 1-6. *CVSS matric groups*

Specifically for assessing severity of vulnerabilities of the security and management engine's firmware, the CVSS is slightly adjusted to better suit the nature of the engine:

- The access vector options used are *network*, *local*, or *physical*. Firmware bugs that can be exploited remotely via the network are more critical. "Local" means that an attacker must access the host operating system with ring 0 privilege in order to mount an attack. "Physical" refers to the capability of reading and/or writing the flash chip that holds the firmware's binary image and nonvolatile data.
- Authentication refers to authenticating to the embedded engine, not the host operating system. Some of the firmware applications, such as AMT, may require user authentication. However, authentication is not required for invoking most of the engine's features from the host operating system.
- Because the engine is a privileged device in the system, the confidentiality, integrity, and availability requirements are all rated at *high* in most cases.

Once a firmware bug is reported, the remediation plan depends on the CVSS score of the bug. The following are the general guidelines:

- If the defect is of low severity, then do not fix or fix in the next scheduled release.
- If the defect is of medium severity, then fix it in the next scheduled release. Prevent firmware downgrade from a firmware version with the fix to any vulnerable version.
- If the defect is of high or critical severity, then fix it in an ad-hoc hot-fix release. Prevent firmware downgrade from a firmware version with the fix to any vulnerable version. If exploitation of the bug may result in leakage of the chipset key or EPID private key, then launch the rekey operation with a remote server after the firmware is updated.

Notice that bug fixes also pose potential risks—they may introduce new functional bugs or security vulnerability, or break working flows. Therefore, complete functional testing and select security reviews should be performed against the fixes for quality control.

Limitations

The CVSS is especially useful for rating software vulnerabilities. However, it is not perfect when used on hardware, in particular because it does not comprehend survivability.

For example, the level of difficulty of patching a hardware bug is not taken into account. The remediation may include the following:

- Documentation and specification change
- Software workaround by remote update
- Firmware workaround by remote update
- Recall (in the worst case)

Such factors should be weighed when evaluating hardware issues.

References

1. Intel IT Center, “Insights on the Current State of BYOD,” www.intel.com/content/www/us/en/mobile-computing/consumerization-enterprise-byod-peer-research-paper.html, accessed on June 10, 2014.
2. Trusted Computing Group, “Trusted Platform Module Library,” www.trustedcomputinggroup.org, accessed on March 20, 2014.
3. Intel, Trusted Execution Technology, www.intel.com/txt, accessed on January 30, 2014.
4. Intel, “Software Guard Extensions Programming Reference,” https://software.intel.com/sites/default/files/329298-001.pdf, accessed on May 10, 2014.

5. CTIA: The Wireless Association, "Smartphone Anti-Theft Voluntary Commitment," www.ctia.org/policy-initiatives/voluntary-guidelines/smartphone-anti-theft-voluntary-commitment, accessed on June 10, 2014.

6. eBay Inc., "eBay Inc. to Ask eBay Users to Change Passwords," http://announcements.ebay.com/2014/05/ebay-inc-to-ask-ebay-users-to-change-passwords/, accessed on June 10, 2014.

7. Target Corp., "Target Confirms Unauthorized Access to Payment Card Data in U.S. Stores," http://pressroom.target.com/news/target-confirms-unauthorized-access-to-payment-card-data-in-u-s-stores, accessed on June 10, 2014.

8. McAfee Labs, "Threat Advisory: EPOS Data Theft," https://kc.mcafee.com/resources/sites/MCAFEE/content/live/PRODUCT_DOCUMENTATION/24000/PD24927/en_US/McAfee_Labs_Threat_Advisory_EPOS_Data_Theft.pdf, accessed on June 10, 2014.

9. Internet Engineering Task Force (IETF), *Request for Comments 6520*, "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension," http://tools.ietf.org/html/rfc6520, accessed on June 10, 2014.

10. OpenSSL Security Advisory, www.openssl.org/news/secadv_20140407.txt, accessed on June 10, 2014.

11. Netcraft, "April 2014 Web Server Survey," http://news.netcraft.com/archives/2014/04/02/april-2014-web-server-survey.html, accessed on June 10, 2014.

12. Kumar, Arvind, Purushottam Goel, and Ylian Saint-Hilaire, *Active Platform Management Demystified: Unleashing the Power of Intel vPro Technology*, Intel Press, 2009.

13. OpenSSL Security Advisory, www.openssl.org/news/secadv_20140605.txt, accessed on June 10, 2014.

14. Intel, Identity Protection Technology, http://ipt.intel.com/, accessed on April 20, 2014.

15. Hewlett-Packard Development Co., "Reducing Security Risks from Open Source Software ," http://h20195.www2.hp.com/v2/GetPDF.aspx%2F4AA0-8061ENW.pdf, accessed on June 10, 2014.

16. National Institute of Standards and Technology, "Advanced Encryption Standard (AES)," http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, accessed on November 17, 2013.

17. National Institute of Standards and Technology, "Security Requirements for Cryptographic Modules," http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf, accessed on April 15, 2014.

18. National Institute of Standards and Technology, Common Vulnerability Scoring System (CVSS), http://nvd.nist.gov/cvss.cfm, accessed on December 12, 2013.