IEEE Access
practical innovations : open solutions

Received December 19, 2013, accepted February 7, 2014, date of publication February 11, 2014, date of current version February 20, 2014.

Digital Object Identifier 10.1109/ACCESS.2014.2305658

Some Fundamental Cybersecurity Concepts

KELCE S. WILSON[1] (Senior Member, IEEE) AND MÜGE AYSE KIY[2]
[1]Patent Litigation, BlackBerry, TX 75094, USA
[2]BlackBerry, Washington, DC 20001, USA

Corresponding author: K. S. Wilson (kwilson@softwareipattorney.com)

ABSTRACT The results of successful hacking attacks against commercially available cybersecurity protection tools that had been touted as secure are distilled into a set of concepts that are applicable to many protection planning scenarios. The concepts, which explain why trust in those systems was misplaced, provides a framework for both analyzing known exploits and also evaluating proposed protection systems for predicting likely potential vulnerabilities. The concepts are: 1) differentiating security threats into distinct classes; 2) a five layer model of computing systems; 3) a payload versus protection paradigm; and 4) the nine Ds of cybersecurity, which present practical defensive tactics in an easily remembered scheme. An eavesdropping risk, inherent in many smartphones and notebook computers, is described to motivate improved practices and demonstrate real-world application of the concepts to predicting new vulnerabilities. Additionally, the use of the nine Ds is demonstrated as analysis tool that permits ranking of the expected effectiveness of some potential countermeasures.

INDEX TERMS Computer hacking, computer security, reverse engineering, software protection.

I. INTRODUCTION

The four concepts introduced here enable comparison and evaluation of protection systems, including both analyzing defeats by known exploits and also predicting likely vulnerabilities. In this section we will introduce these concepts which will be expanded in the sections that follow.

These concepts resulted from one of the authors' participation in an early test and evaluation program run by the U.S. Department of Defense (DoD) [1]. The DoD engaged penetration testers (a.k.a. ‘‘white hat hackers’’) to attack its own computer protection systems and report back the results of their attempts, with the intent that such testing would expose weaknesses so that protections could be strengthened. In the early 2000s, the DoD sought an explanation for how both the protection experts and the white hat hackers could simultaneously claim victory in the same testing project (i.e., the same hacking test).

The explanation is disturbing due to the potential that the situation is spread across the cybersecurity community: the protection experts and hackers each defined victory differently. Protection experts defined victory as preventing hackers from completing the specific attack vectors against which the protections ostensibly defended, whereas hackers defined victory as being able to complete at least some serious attack vectors that they believed had injured the integrity of the computing asset. In the early stages of the testing program, the protection experts and the white hat hackers had each been focusing on different attack vectors.

As a result, a significant discovery was made: Each of the ‘‘protected’’ computing assets had retained notable vulnerabilities, despite being labeled as ‘‘protected’’ by experts. Different protection systems left a different set of vulnerabilities intact, but all protection systems that were available in the commercial marketplace left at least some. Although individual experts may have been quite competent in addressing a particular set of attack vectors, it was apparent that different experts had been concentrating on different sets, without coordination to ensure completeness across all potential attacks. There was some degree of overlap, but not a common set of test vectors against which all experts had been evaluating proposed protection systems.

Upon realization of this situation, and its gravity, the claims of victory by the white hat hackers in the early tests were analyzed for commonality. This differentiated security threats into distinct classes, producing **three threat classes** that will be discussed in the next section: **piracy**, **tampering**, and **reverse engineering** [2]. A matching set of protection categories was then identified: license enforcement, anti-tamper, and anti-reverse engineering. The test planning efforts then leveraged this categorization in subsequent test projects to analyze various available security products for effectiveness against each different threat class. Another significant discovery was then made. **There was no one-size-fits-all protection system available in the commercial marketplace.**

This meant that an effective cybersecurity program would require additional work. The owner of a computing asset would need to ascertain all threat classes against which a defense was desirable. Only then could the proper security product be selected for use, and in many situations, more than a single security product would be needed. This added a complicating factor. Not only would each security product require evaluation for effectiveness against each threat class for which protection was advertised, but each product would also require evaluation for compatibility with other products that might be used contemporaneously.

Additionally, it became obvious that a protection system that was implemented using only programming techniques within application software could be defeated by attack tools that intercepted calls from the software to the operating system (OS) and falsified or altered the outgoing or incoming data [3]. This type of vulnerability persisted, even if the protection system had been designed well enough to provide a solid defense against software-based attacks, such as the use of an application-layer debugger. The idea of ''tunneling under the castle wall'' – effective in medieval warfare – turned out to be similarly effective in cyber warfare. Essentially, a protection system could only be reliably effective against attacks that occurred at the same system layer in which the protection system had been implemented, or attacks at higher layers.

An easily-understood example of ''tunneling under'' a protection system is the use of virtual machines and other tools to perform execution run tracing and data falsification. These tools can force many protections to reveal secrets that are relied upon for security. Once the secrets are revealed, protections that rely upon those secrets for security can be defeated. Thus, even if well-designed, protections can be vulnerable to attacks from lower layers. A definition of computing layers is needed that enables meaningful analysis of whether an attack is at the same layer as a protection system, beneath it, or above it.

The common privilege ring model for computer security [4] does not reach with sufficient refinement into hardware-based protections, and the TCP/IP stack model for networked computing systems [5] is not sufficiently tailored to security issues to properly model protections and attacks. A **new five layer model** was developed that reaches downward into separate hardware layers and upward into software layers.

To focus risk mitigation efforts against a class of attack that includes code lifting, a **payload vs. protection paradigm** is introduced to explain how program functionality can be separated from protections. In a code lifting attack, some protections are not so much defeated as merely sidestepped. Such attacks use a two-phase plan of:

1. Separating desirable portions of the computing asset from at least some of the protections – to produce a less-protected version – and then
2. Attacking the less-protected version [2], [6].

This is more feasible when the computing asset is software rather than a hardware device, and the protection system is software-only.

Finally, a collection of best practices is proposed as the **nine Ds of cybersecurity**. Although the nine Ds are not a comprehensive list, they do include both the three tenets of cybersecurity proposed by the DoD [7], [8], as well as lessons-learned about implementation flaws introduced by human error. The nine Ds are presented in a manner that is designed to be easily remembered by security system implementers.

The concepts here should facilitate categorizing security products by the protection offered, rating the products' effectiveness within each threat class, analyzing breaches of existing protections, and predicting likely vulnerabilities of proposed protection system designs. By iterating analyses of proposed designs and addressing predicted vulnerabilities, more effective protection system designs can be achieved.

II. THE THREE SECURITY THREAT CLASSES

The key to achieving effective protection system design is developing a strategy based upon an analysis of relevant threats [2], [7], [8]. A good strategy will counter all relevant threats ensuring adequate coverage of each threat class, rather than merely using whatever technologies that happen to be available. The process should begin by ascertaining the entire set of threats that are of concern to the owner of a computing asset and then analyzing the degree of each threat. Computing assets may be hardware or software, including application program software, networks and solo computing systems intended for either secure site operation or mobility in uncontrolled environments. As used here, the term computing system refers to a combination of hardware and software that is necessary to make use of that hardware.

A classification system is proposed for the threat classes; (1) piracy, (2) tampering, and (3) reverse engineering.

There are varying degrees of intensity within each threat class, due to varying levels of hacker capability and computing asset value. The variations in hackers' capabilities are described in the section on the nine Ds. It should be noted that there will be differences in specific relevant threats for application security versus network security scenarios, although the general concepts introduced here are relevant to both. See page 38 of [2]. In some situations, certain attacks may not be an issue; authors of freely-distributed programs may not worry about piracy.

Fig. 1 provides a representation of a threat environment, illustrating attacks against a computing asset from three different directions, labeled as piracy, tampering, and reverse engineering. These will be discussed in turn.

Piracy is defined here as unauthorized use; it takes many forms. These include distributing an excessive number of software copies, moving a copy to an unauthorized location, an excessive number of different people using a single copy, using a copy beyond a specific date or a specified number of trial uses, and using supposedly prohibited features (for example, saving data in certain file formats).

The counter to piracy is license enforcement; examples include node-locking and hardware components such as

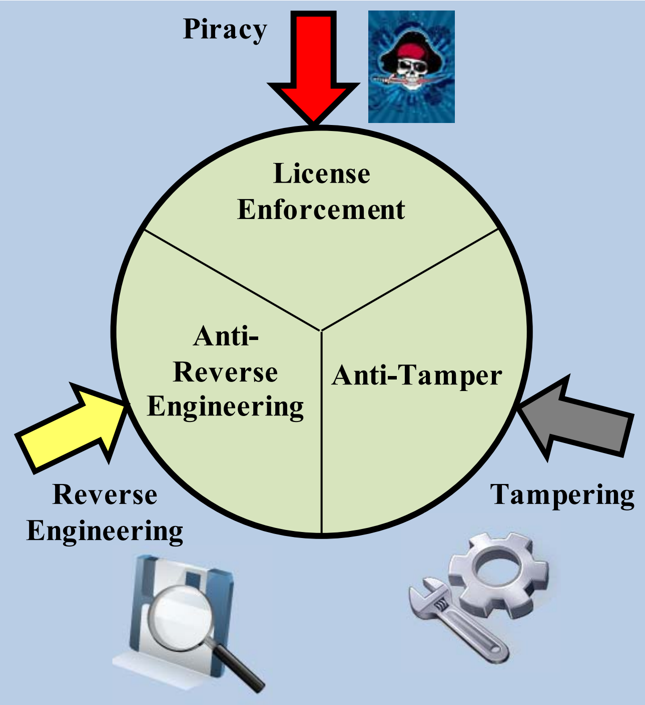

FIGURE 1. Graphical depiction of threat classes and protection categories.

dongles that are more difficult to replicate than merely copying software.

The second attack class is tampering: unauthorized alteration of a computing asset. There are multiple types of unauthorized alteration, including altering functionality or capability, introducing malicious logic, and disabling or modifying security controls. Examples include computer viruses and introduction of ''Easter eggs'' and ''backdoors'' (a.k.a. ''trapdoors'') by malicious software developers [9].

Functionality changes can take different forms, such as degrading software capability to deprive authorized users of proper functionality and increasing capability for improper use by others. Introduction of malicious logic can also take multiple forms; a virus can migrate to remote sites and a programmer can covertly introduce malicious logic during development. Developer-inserted malicious logic occurs for reasons such as revenge and to facilitate future attacks.

Some common anti-tamper protections against network attacks are firewalls and anti-virus products, with firewalls providing prospective protection and anti-virus products providing remedial protection. Anti-tamper protections for application software include secure loaders, integrity checks, self-healing that can automatically reverse some hacks, and the use of a secure development environment (SDE) to protect against malicious or careless programmers.

The third attack class is reverse engineering, which involves learning how a program operates and can be used for stealing intellectual property (IP). IP theft can provide a competitive advantage if an attacker (a.k.a. ''hacker'') can inexpensively learn trade secrets that are buried in a program's functionality. This can be an acutely painful economic problem when the original developer had spent considerable expense developing and refining the ideas implemented in the program.

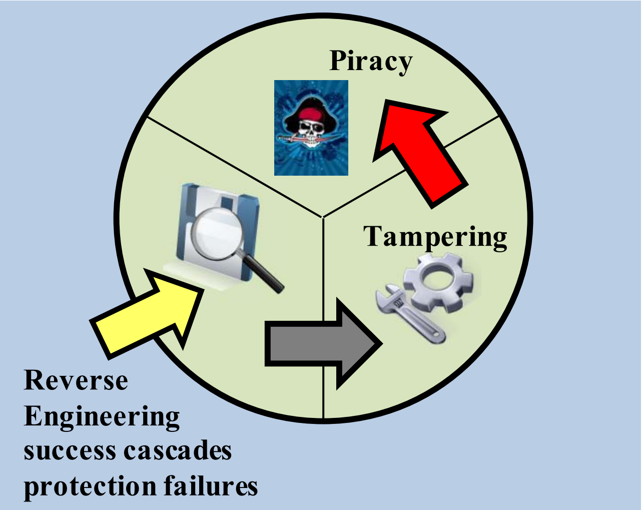

FIGURE 2. Graphical depiction of an attack on a computing asset.

Reverse engineering is also often used as a first step in defeating protections [10]. A graphical depiction of this is given in Fig. 2, illustrating how reverse engineering eases tampering, which then permits piracy. Examples of this approach include an attacker identifying specific protection algorithms used, the location of encryption keys, and the memory addresses of critical functionality. Common attack methods include profiling behavior, performing run traces, and disassembling and decompiling an executable binary file.

Anti-reverse engineering protections include encrypting the executable file and performing code obfuscation, whether of the source code, the binary executable, or both.

Tampering may support reverse engineering when an attacker makes targeted alterations to data or an executable file and correlates those alterations with observed behavior changes. Iterating observations with successively better-targeted tampering can permit incremental advances in the reverse engineering effort. Thus, anti-tamper and anti-reverse engineering protections may be complementary.

One way to leverage synergy is to design anti-tamper protections, such as self-healing, to prevent the disclosure of information that permits attackers to learn which alterations work toward defeating other protections. Even license enforcement can assist with a mutual defense by reducing propagation of computing assets, reducing the exposure to additional skilled attackers in different locations.

Unfortunately, many defensive protections are effective in only a single threat class or against only some of the attack vectors within a single threat class. Some protections, however, offer broad protection within a first threat class and a lesser degree of protection against a second threat class – perhaps effectiveness against only a small set of attack vectors within that second threat class. This is because the different classes of threats are so disparate that it is unlikely a single defensive measure can adequately address all attack vectors within all of the threat classes.

In general, protections should be integrated to the fullest extent that is practical. A well-engineered combination can produce synergistic results when some of the protections enhance the effectiveness of others. Conversely, a poorly-engineered combination can undercut effectiveness when a failure of one protection measure facilitates attacks against another protection measure.

Well-designed protection systems should defend against all relevant attack vectors and may employ multiple protection measures to cover multiple threat classes, multiple threats within each class, and multiple layers.

III. THE FIVE LAYER MODEL

It is useful to model computing systems as comprised of multiple layers to facilitate analyses. Well-known examples are the TCP/IP stack model that currently has variations in four, five, and seven layers, and also the four layer privilege ring model. A five layer model, illustrated in Fig. 3, is based on straightforward groupings of observable attack vectors.

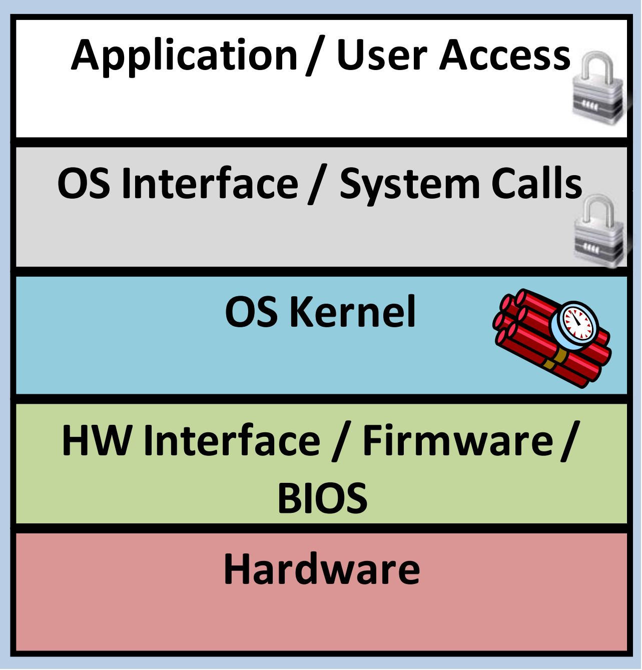
Application / User Access
OS Interface / System Calls
OS Kernel
HW Interface / Firmware / BIOS
Hardware

FIGURE 3. **Five layer model illustrating a flawed layer 3.**

The proposed five layer system is

1. Application layer/ User Access
2. OS Interface / System Calls
3. OS Kernel, OS primary functionality
4. HW Interface / Firmware / BIOS, boot kernel
5. Hardware (HW), CPUs, memory, interposers

In general, protections can be implemented at any layer and protections will be needed at each layer. In the Fig. 3 notional system, the two top layers have protections. A vulnerability, represented as a time bomb, is in a lower layer: the OS kernel. This presents an opportunity for an attacker to ‘‘tunnel under’’ the protections.

In some cases, application security might be software-only. Even if it had been well-designed and implementation was flawless, some vulnerability in the lower layer OS kernel could be used to defeat the application layer protections. For example, consider the case of an anti-tamper protection that performs self-checking and automatic repair of critical data and instructions. Such protection can be implemented purely within software and will enable an application to check its own integrity and make repairs. However, if the OS kernel has been compromised, the self-checking might be redirected to a different memory location.

In this scenario, two copies of the application program will be loaded into memory. One remains intact while the other executes in a tampered state. When the executing copy performs a self-check, the flawed OS can redirect the self-check mechanism to the intact copy. Alternatively, if the self-check uses checksums for integrity determination, the OS can place forged checksum values into critical memory locations. The self-check protection will operate with an incorrect determination, and there will be no repairs.

The following should govern the use of the model:

- Even ideal protections with perfect implementation can potentially be defeated by lower layer attacks.
- A vulnerability at one layer can create one above.
- A vulnerability can negate lower layer protections.
- Protections should be implemented at each layer.
- Protections at different layers should be integrated.

An example of using the five layer model concept is the BlackBerry security system. BlackBerry integrates design of hardware, OS, and applications on the mobile device itself with infrastructure elements and management controls to create an integrated security solution [11]. The effectiveness of this integrated approach has been at least somewhat validated by the US DoD’s action of forcing some of its employees to return to BlackBerrys and give up iPhone and Android devices [12].

Integration of protection systems at different layers might provide synergistic effects, similar to those noted for protections against different threat classes. Even beyond integrating protection systems with each other, there may be a further need to integrate protection systems with the actual functionality that gives the computing asset its value.

IV. PAYLOAD VS. PROTECTION

A computing asset may be viewed as a combination of two parts: (1) the functionality that gives an asset value (i.e., ‘‘payload’’), and (2) the measures that the asset owner puts in place to control its use (i.e., ‘‘protection’’). The payload is the functionality that is available to authorized users and the protection is the set of features that ensure both trustworthy operation of the functionality and also that the functionality is available only to authorized users. The terms ''hacking'' and ''cracking'' (see p. 37 of [2]) refer to attempts to access functionality in violation of the protection.

An attacker can attempt to access a payload by removing or altering protections. This may be easier when a protection is ''bolt-on'' and operates independently from payload logic. Some ''bolt-on'' security products are competently designed for both effectiveness and ease-of-use. Unfortunately though, if security is easily tacked onto an existing software package, it might be just as easily separated as illustrated in Fig. 4.

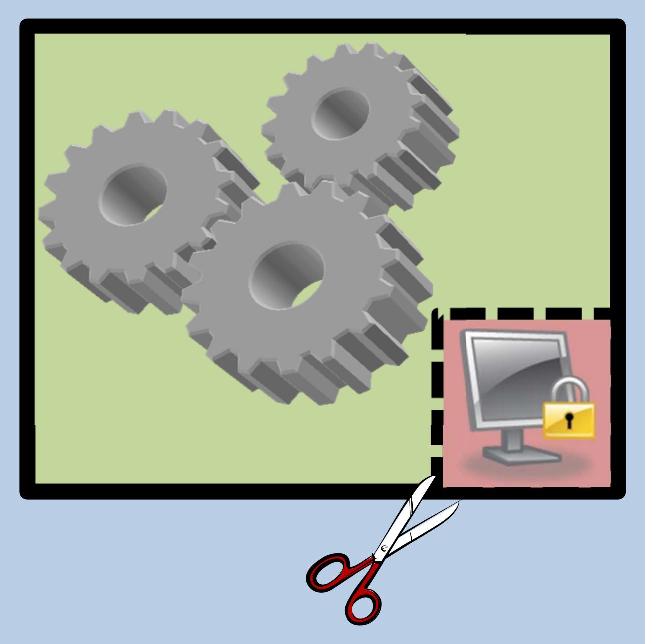

FIGURE 4. Graphical depiction of the payload vs. protection paradigm.

One example of an attack that separates payload from protection is code lifting [2]–[6]. It is a two-phase attack that can sidestep at least some protections rather than attempting to defeat them outright. First, valuable functionality is copied from a binary file that comprises an executable module or a critical data set of an ostensibly protected program. Next, this functionality is then placed into a shell program that provides the proper execution environment. Although creating the shell program does require effort, the level of effort may be lower than either the effort required for independent development or defeating in-place protections.

Ideally, a comprehensive cybersecurity strategy should be implemented from the very beginning of a project so that protections can be thoroughly integrated. However, bolt-on type security may have cost and schedule advantages. An example of a bolt-on protection is an encryption wrapper that operates on an executable program to produce a combination product: a secure launcher and an encrypted program that is decrypted just in time for execution and is deleted after execution completes. For such systems, the protection step may have less impact on development because validation of the payload functionality can be completed prior to final security work. The post-security testing merely needs to ensure that the protection system did not damage functionality. Attacks against such protections include a memory grab that attempts to copy the decrypted executable and store it in an executable state. If protections had been put into the executable portion of the program, the memory grab or other code lifting attack might bring the protections along.

If the integration is sufficiently thorough, it may be difficult for an attacker to distinguish between desirable functionality and the protection logic. The attacker may need to purchase a copy of the same security product that had been used, and apply it to a test program.

Differential analysis performed on the attacker's original and protected test programs might provide insight into the security product's signature and facilitate later attacks [13]. Protections found by this signature exploitation method might be defeated simply by jumping over or replacing the protection instructions with no-operation commands (NOPs).

A strategy for preventing the separation of payload and protection is to integrate protection measures so thoroughly within the payload's core functional logic that the process of separation becomes too difficult. One approach is to weave protection and payload logic such that altering protection logic will also damage payload logic to the point that the payload loses its value to the attacker. Unfortunately, this may be impractical; time and budget constraints may dictate a more modest plan. Cybersecurity requires a balancing act between many factors, and cost is one of them [14].

V. THE NINE Ds OF CYBERSECURITY

Using the nine Ds proposed here can help achieve a decent balance. They provide easily-remembered guidelines, and are inspired by the DoD's three tenets of cybersecurity [7], [8], which are:

1. Focus on what is critical;
2. Move critical access points ''out of band;'' and
3. Detect, React, Adapt.

The DoD proposes that protection systems should have characteristics of feasibility, adoptability, and sustainability. The focus of protections should be to reduce:

1. System susceptibilities,
2. Access to potential system flaws by hostile parties, and
3. Capacity of hostile parties to exploit system flaws.

A. HISTORY AND SIGNIFICANCE OF THE NINE Ds

One of the white hat hacking tests mentioned in the Introduction section involved a computing asset that had been protected by a commercially available security product that incorporated patented technology and was the subject of multiple peer-reviewed academic articles. The academic work ''proved'' that the underlying protection theory was secure. In preparation for the test, something on the order of 100 engineering hours, using the protection company's best technical experts, were spent tailoring the application of the security product to the computing asset.

The result? ***Defeat within a mere seven minutes*** by one of the authors of this article [2].

This type of thing was, unfortunately, quite common: academic theories failing completely in real-world testing. So, in addition to analyzing defeats in order to classify them according to attack types, computing system layers, defeated versus sidestepped, another analysis was performed: was the defeat attributable to an inherent weakness in the underlying theory, or was it instead human error in the implementation?.

In the test just mentioned, the defeat of the system was a result of human error, rather than the underlying theory. The engineers had access to journals and articles, but the body of academic work was useless to prevent the protection failure.

Human error occurs despite the availability of a plethora of literature on some topic. It occurs because the engineers and technicians implementing a system can individually track only a limited number of concepts simultaneously, and communication among team members also has limits.

What was needed to reduce the risk that human error would degrade a protection system's effectiveness was a focus aid – something to assist planners and implementers with simultaneously tracking more of the relevant concepts reliably. An analysis of the human errors led to a valuable insight: Most of the errors were a result of violating one of the DoD's three tenets or failing to heed one of the concepts introduced here.

A list of points to remember was compiled and each of the memory points was (somewhat contortedly) associated with a word that began with a common letter: D.

The nine Ds proposed here should be easy to remember and thus assist with reducing the occurrence of embarrassing human error.

B. DETER ATTACKS

An attacker needs three things for a successful attack:

1. The will to attempt an attack;
2. The ability to succeed with the attack; and
3. Access to the system.

Deterrence measures are those that work to reduce an attacker's will to attempt an attack, such as threats of legal action or other punitive measures. Unfortunately, deterrence is prone to failure. Protections that work against ability are technical protection measures (''TPMs''). Curtailing system access can often be accomplished through policies.

C. DETECT ATTACKS

Detection of malicious activity is necessary if affirmative reactions will be part of the defensive strategy. One example is a password fail counter that triggers a memory wipe upon the counter reaching a threshold value. Another is a ''phone home'' system that reports attacks to a monitoring location.

Other detection systems might include behavior monitors that watch for indications of compromise. For example, an excessive amount of data traffic might indicate that a running program has been altered to forward data to a remote site.

D. DRIVE UP DIFFICULTY

Driving up difficulty often involves the use of TPMs to make attacks more expensive. Attackers can be defeated by driving the level of difficulty beyond their ability to cope.

Attacker skill can be stratified in five basic levels:

1. A Script Kiddy can only perform pre-fabricated attacks that were prepared by someone else.
2. A Novice can create new attacks with existing tools.
3. An Expert can create more capable attack tools, but may face resource constraints when operating alone.
4. Funded Organizations are groups of experts that can develop sophisticated attacks based on novel tools.
5. Nation States can access world-class expertise and are effectively free from resource constraints.

Organized crime groups that steal banking credentials for large-scale theft projects are funded organizations. In general, no protection system should be considered immune from possible defeat by a nation state.

Three considerations for analyzing difficulty are [7], [8]:

1. Inherent system weakness;
2. Attacker (hacker) access to the weakness; and
3. Attacker (hacker) capability to exploit the weakness.

Attacker capability is something over which a protection specialist has no control. As new attack tools are developed, attacker capability will increase over time. This leaves only the first two considerations, weakness and access, for driving up difficulty. The options are to reduce inherent system weakness and restrict availability to reduce attacker access.

In addition to skill level, it is also possible to classify attackers as either rational or irrational. A rational attacker performs a cost/benefit analysis and proceeds only if the ratio is favorable. Cost is measured not only with currency, but also with time and other resource demands. For some computing assets, the benefit of a successful attack can be very high. For example, if protected IP has national security significance or high competitive value, a successful exploit could be worth millions of dollars. Benefits can be valued by some attackers as more than merely monetary. There may be an element of emotion involved, such as ego or a desire for revenge. To defeat rational attackers, a protection system need only present a sufficient level of difficulty to render the attackers' perceived cost/benefit ratio unfavorable.

In contrast, an irrational attacker will proceed regardless of the perceived cost or benefit. Irrational attackers are unlikely to be deterred, and can often be stopped only by insurmountable difficulty or insufficient access. Fortunately, the more highly-skilled attackers are likely to be rational.

E. DIFFERENTIATE PROTECTIONS

As mentioned in the section titled Three Security Threat Classes, protection systems should each be focused on one or more specific threats that had already been identified. This requires (1) ascertaining each proposed system's likely performance against the identified threats, (2) ensuring that interactions among various proposed contemporaneous systems do not hinder performance goals, and (3) ensuring that there is an acceptable level of expected resilience against each threat.

F. DIG BENEATH THE THREAT

As mentioned in the Five Layer Model section, an attack at a layer that is lower than a particular protection may be able to defeat that protection – even if that protection is perfectly implemented. It follows then, that a protection at a lower layer than an expected attack may be able to defeat the attack – even if that attack is expertly conducted.

G. DIFFUSE PROTECTION THROUGHOUT THE PAYLOAD

As mentioned in the Payload vs. Protection section, protections can be integrated throughout a payload's core functional logic to drive up difficulty for code lifting attacks. The goal of this D should be to force the attacker into a choice: either bring along functioning protections or else forfeit the payload value.

H. DISTRACT WITH DECOYS

Attackers will stop either when they become frustrated or when they believe that they have succeeded. Encouraging a false belief in success is a valid protection option.

I. DIVERT ATTACKERS TO OTHER TARGETS

Another effective strategy is to divert attention to a more attractive target elsewhere. The well-known adage about not needing to outrun a bear, if you can outrun one other person, can also apply to cybersecurity. You could ''win'' merely by persuading an attacker to target someone else.

J. DEPTH OF DEFENSE

The concept of defense in depth is a valuable one for use against sophisticated attackers [15]. A real-world, practical implementation is that only after an attacker has defeated some protections will other protections manifest themselves. The value in this is that the second and higher tier defenses are not exposed for study by lower skill attackers who lack the resources, will, or access to defeat the first tier defenses.

One example of this is the use of a secure launcher that performs just-in-time decryption of the primary executable program that, in addition to being encrypted in its resting state, also uses obfuscation and self-healing. In this example, the first tier of protection is the encryption of the binary executable, which keeps the binary from being edited with a hex editor. This first tier of defense is useful against static attacks, which are aimed at an executable as it resides on permanent media. The secure launcher can be leveraged to prevent static attacks and make dynamic attacks more difficult. Dynamic attacks are those that are implemented against a running program.

The use of the secure launcher can offer protection by

1. Detecting the presence of attack tools on the system,
2. Decrypting the primary executable only after determining the environment to be safe from these attack tools,
3. Monitoring for malicious activity, such as memory grab attempts, and
4. Ensuring that upon completion of execution the memory and hard drive swap space are cleared of residual data and instructions.

Obfuscation protection is thus only visible to an attacker after the encryption protection has been defeated, for example through a memory grab or a run trace. Only then, after the obfuscation has been sufficiently defeated to enable an attacker to identify critical sections for tampering, will any self-healing defenses manifest themselves. Hopefully, the encryption and obfuscation will prevent identification of self-healing defenses prior to their activation.

VI. DISTURBING CONSIDERATIONS FOR MOBILE TELECOM DEVICES: REAL-WORLD APPLICATION OF THE CONCEPTS

Smartphones can't always protect not-so-smart users.

Mobile telecommunications (''telecom'') devices, such as smartphones, present some significant security challenges. Due to their mobility and small size, they have a high potential to be lost or stolen, and thus fall into hostile hands. In such a scenario, a mobile telecom device is under the physical control of an attacker. The control may be permanent, perhaps for the purpose of learning valuable information such as bank account numbers or possibly copying sensitive documents. Alternatively, the control may be temporary, to surreptitiously insert malware and return the device to the unwitting owner, who then unknowingly discloses private information at a later time. To combat these threats, the devices need to protect themselves without assistance, similar to the paradigm for application-centric security practices. For some devices, though, assistance may be available in the form of a remotely-initiated data wipe.

Additionally, because mobile telecom devices are complex computing platforms that operate on public networks, a comprehensive security plan must also include network-centric security practices. An attacker may attempt to access a device remotely, through Trojan horse malware or breaking in through a poorly-protected wireless port.

Combining the factors of high mobility, small form factor and computational power, with usage in cellular, WiFi, and Bluetooth networks, we see that mobile telecom devices may be facing one of the worst possible environments for security threats. The emergence of self-assembling ad hoc networks will present a new generation of security challenges. Comprehensive security programs, based on the concepts introduced here, can help to mitigate some of the risks.

Unfortunately, though, many users happily load spyware onto their own devices. They do this with the mistaken belief that the harmful programs are merely innocent applications – either fun games or valuable utility software [16]. One way to address this risk is by limiting software installation to only those programs that have been subjected to a security review and endorsed with a code signing certificate.

Also, there is a potentially catastrophic vulnerability that currently exists in telecom devices – likely including the one that you are using right now. Smartphones and notebook computers can be converted into covert eavesdropping and tracking devices to spy on their users, even when the users believe that the devices have been turned ''off'' [17]. This can occur because (1) the shutdown procedures are logic-controlled, and (2) ''off'' isn't really ''off'' in the traditional sense that power is no longer being supplied to processing circuitry. Even when mobile telecom devices are turned ''off'' by their users, some software-implemented functionality, such as alarm clocks and other timers, continues to operate. Thus, the device often isn't truly powered down to a thoroughly non-operable state.

When contemplating that shut-down procedures are controlled by logic, a frightening threat scenario emerges: Malware might alter logic that controls a shut-down procedure so that a device merely appears to be ''off'' by darkening the screen and managing other observable behavior. This altered state can deceive the user, even as the device surreptitiously continues to operate one or more of the microphone, camera, or GPS or other sensor systems. Collected information can then be offloaded at a time and in a manner that is likely to escape detection by the user.

Had you already been aware of this risk, or have you been exposing yourself to it?

Note that, despite the plethora of theoretical academic work on cybersecurity topics, major threats continue to exist entirely unnoticed. However, this threat is easily understood in light of the practical concepts introduced here.

It is also possible to rate the potential effectiveness of proposed and available countermeasures using the nine Ds and the five layer model. For example, one effective countermeasure (apart from the obvious of not having any telecom device in the vicinity of a sensitive conversation) is to remove the battery. This is a solution in the lowest layer (hardware). and ''digs beneath the threat'' to provide a robust solution, as suggested by the nine Ds. Another hardware-based solution is to place the mobile device in a box or holster that blocks light, sound and wireless signals.

Protection systems for mobile devices that exist only in the upper layers are inherently less secure. For example, Samsung's Knox product and Good Technology's mobile device management (MDM) system have both been shown to be insecure as a result of Android OS vulnerabilities [18], [19]. Both of these real-world examples fit the situation illustrated in Fig. 3. It is important to note that because the OS is lower in the five layer model – and not within the control of either company – security risks will likely persist no matter how much effort is put into enhancing upper layer software.

In contrast, a mobile device implementing a protection solution that comprehensively integrates security from hardware up through the OS and into software layers, and uses code signing and integrity verification of critical device control logic can be expected to provide superior resistance to attack. Nevertheless, because a malicious logic attack against shut-down procedures would occur in one of the layers one through four, it is clear that a hardware solution is a strong contender for a solid defensive plan. One such solution is the use of a mechanical switch that has no logic bypass and physically removes power or otherwise disables one or more sensors and/or wireless operability.

Using the concepts introduced here not only makes it easy to predict and understand the attack itself, but also enables evaluation of proposed solutions and nearly instantaneous identification of where potential defensive measures could be implemented.

VII. SUMMARY

Four important cybersecurity concepts were described in the context of lessons-learned from successful hacking attacks against cybersecurity protection tools that had been touted as ''secure'' by protection experts. The concepts explain why trust in those systems was misplaced and are: (1) differentiating security threats into distinct classes; (2) a five layer model of computing systems; (3) a payload vs. protection paradigm; and (4) the nine Ds of cybersecurity.

The value of the concepts was illustrated by examining a disturbing cybersecurity threat that is easily understandable and nearly trivially predictable in light of these concepts – but yet had previously been largely overlooked or unknown.

REFERENCES

[1] (2014, Feb. 14). *Software Protection Initiative* [Online]. Available: http://spi.dod.mil

[2] K. Wilson, ''An introduction to software protection concepts,'' *Intell. Property Today*, vol. 14, no. 8, pp. 36–41, Aug. 2007.

[3] Z. Wang, X. Jiang, W. Cui, and P. Ning, ''Countering kernel rootkits with lightweight hook protection,'' in *Proc. 16th ACM Conf. Comput. Commun. Security*, Nov. 2009, pp. 545–554.

[4] G. Duarte. (2008, Aug.). *CPU Rings, Privilege, and Protection* [Online]. Available: http://duartes.org/gustavo/blog/post/cpu-rings-privilege-and-protection

[5] P. Simoneau. (2006). *The OSI Model: Understanding the Seven Layers of Computer Networks*, Global Knowledge, Cary, NC, USA [Online]. Available: http://faculty spokanefalls.edu/Rudlock/files/WP_Simoneau_OSIModel.pdf

[6] G. Balakrishman, R. Gruian, T. Reps, and T. Teitelbaum. (2005). *CodeSurfer/x86—A Platform for Analyzing x86 Executables* [Online]. Available: http://research.cs.wisc.edu/wpis/papers/cc05-tool-demo.pdf

[7] J. Hughes and G. Cybenko, ''Quantitative metrics and risk assessment: The three tenets model of cybersecurity,'' *Tech. Innovation Manage. Rev.*, pp. 15–24, Aug. 2013.

[8] (2014, Feb. 14). *The Three Tenets of Cyber Security* [Online]. Available: http://spi.dod.mil/tenets.htm

[9] F. Greitzer, A. Moore, D. Cappelli, D. Andrews, L. Carroll, and T. Hull, ''Combating the insider cyber threat,'' *IEEE Security Privacy*, vol. 6, no. 1, pp. 61–64, Jan./Feb. 2008.

[10] D. Quist and Valsmith. (2007). *Covert Debugging Circumventing Software Armoring Techniques* [Online]. Available: https://www.blackhat.com/presentations/bh-USA-07/Quist_and_Valsmith/Whitepaper/bh-USA-07-quist_and_valsmith-WP.pdf

[11] (2014, Feb. 14). *BlackBerry security* [Online]. Available: http://us.blackberry.com/business/topics/security/overview.html

[12] A. Sternstein. (2013, Dec. 3). *Pentagon Disconnects iPhone, Android Security Service, Forcing a Return to BlackBerry for Some* [Online]. Available: http://www.nexlgov.comlrrobiIel20131121defense-disconnecls-iphone-android-security-service-forcing-return-blackberry-sorne/747531

[13] S. Garfinkel, A. Nelson, and J. Young. (2012). *A General Strategy for Differential Forensic Analysis* [Online]. Available: http://www.journals.elsevier.com/digital-investigation

[14] K. Wilson. (2011, May, 24). *How Much Should You Invest in Software Security*, AFCEA-GMU C4I Center, Fairfax, VA, USA [Online]. Available: http://c4i.gmu.edu/events/reviews/2011/papers/4-Wilson-paper.pdf

[15] (2014, Feb. 14). *Defense in Depth* [Online]. Available: http://www.nsa.gov/ia/_files/support/defenseindepth.pdf

[16] Fox News. New York, NY, USA. (2014, Jan. 27). *NSA Spying Through Angry Birds, Google Maps, Leaked Documents Reportedly Reveal* [Online]. Available: http://www.foxnews.com/tech/2014/01/27/nsa-spying-through-angry-birds-google-maps/

[17] K. Wilson, ''Conflicts among the pillars of information assurance,'' *IEEE IT Prof.*, vol. 15, no. 4, pp. 44–49, Jul./Aug. 2013.

[18] Cloud Security Alliance. Dallas, TX, USA. (2013, Apr. 25). *How Secure is Mobile Device Management Anyway* [Online]. Available: https://blog.cloudsecurityalliance.org/2013/04/25/how-secure-is-mobile-device-management-anyway/

[19] J. Cheng. (2014, Jan. 10). *Samsung: Knox Security Gap Not Specific to Galaxy Devices* Wall Street Journal Digits, New York, NY, USA [Online]. Available: http://blogs.wsj.com/digits/2014/01/10/samsung-knox-security-gap-not-specific-to-galaxy-devices/

KELCE S. WILSON received the B.S. degree in electrical engineering from the University of Arizona, Tucson, the M.B.A. degree from Chapman University, Orange, CA, USA, the M.S. and Ph.D. degrees in electrical engineering from the Air Force institute of Technology (AFIT), Wright-Paterson Air Force Base, OH, USA, and the Juris Doctor (law degree) degree from Capital University Law School, Columbus, OH, in 1988, 1993, 1994, 1998, and 2005, respectively. He received the U.S. Patent and Trademark Office patent practitioner registration in 2002, and a law license from the Texas bar in 2005.

He was commissioned as an officer with the U.S. Air Force in 1988 and served on active duty until 2002, when he transferred to reserve duty. He retired from the U.S. Air Force Reserves as a Lieutenant Colonel in 2010. During his time with the Air Force, he was involved in satellite control, military cryptographic systems, radar jamming, low-observable (stealth) aircraft, automatic target recognition, synthetic aperture radar, and cybersecurity testing.

Dr. Wilson was given a commendation by the commander of the National Geospatial-Intelligence Agency in 2005 for his work as an Adjunct Professor teaching radar courses to analysts within multiple U.S. National Intelligence Agencies. He was named Reservist of the Year by the Air Force Aeronautical Systems Center in 2004 for solving multiple intractable mathematical problems affecting proposed designs and modifications of operational radar jamming and stealth aircraft systems.

MÜGE AYSE KIY received the B.S.F.S. degree in international politics from Georgetown University, Washington, DC, USA, and the Juris Doctor degree from the George Washington University Law School, Washington, in 2007 and 2011, respectively. She received a law license from the New York bar in 2012 and from the District of Columbia bar in 2013.

From 2007 to 2010, she was an Assistant with BlackBerry Corporation's Government Relations Team. From 2010 to 2011, she was a Policy Specialist, and since 2011, she has been a Federal Lobbyist with BlackBerry Corporation, Washington, DC. In her current role, she is responsible for maintaining the relationships between BlackBerry and the agencies of the Executive Branch. In addition, she handles the company's policy and advocacy strategies at the state level while serving as the U.S. policy lead on several issues including Privacy, Cybersecurity, Health IT, and Distracted Driving.

Ms. Kiy is a member of the New York State Bar Association, District of Columbia Bar Association, and the Federal Communications Bar Association.