TEN REQUIREMENTS FOR SECURING FINANCIAL TRANSACTIONS AND CRITICAL INFRASTRUCTURE ACROSS INSECURE NETWORKS

Martin Quiroga and Dr. Michael Fiske
maq@biogy.com
mikef@biogy.com

Presented April 27, 2009 at the
2009 NSA / Department of Defense
Identity Protection & Management Conference
Edited September 7-14, 2010

INTRODUCTION

Fraud, identity theft and data theft are rampant and continue to increase at alarming rates in spite of the enormous amounts of technology and resources that institutions and governments devote to combating these crimes. Moreover, it is clear from the tremendous and wide-scale damage inflicted by recent worms, such as Conficker, that existing state of the art identity protection and management (IPM) systems fall short of containing or minimizing the spread of successful cyber attacks used for committing these crimes.

The core of the problem lies in the shortcomings of vulnerability assessment and remediation procedures. First known vulnerabilities must be detected (i.e. an unpatched web server) and subsequently patches for the affected operating systems or applications must be applied as updates. These procedures are tedious and can often affect operational continuity, which means they are often delayed or altogether neglected.

Further aggravating this problem is the highly-mobile nature of today's workforce, and their dependence on a portable PC. This significantly complicates the logistics of administration and management of the portable PC, which are often connected in highly hostile and unprotected environments, such as airports or coffee shops. Thus the critical tasks of vulnerability assessment and remediation are left to end-users -- arguably the least equipped individuals to deal with such endeavors. Hence, a "perfect storm" environment is created for wide-scale attacks, such as worms.

It was recently discovered that the U.S. electrical grid had been compromised by spyware, most likely planted by Chinese or Russian hackers[1]. The level of sophistication involved in this incident as well as in software code such as Conficker and other malware suggests that well-financed organizations and governments sponsor these developments.

Present-day estimates of the economic impact of cyber crime exceeds a staggering $US 1 trillion dollars per year[2]. Significantly contributing to this monetary figure are preventable attacks that leverage previously obtained information such as static login tokens, credit card verification parameters and cryptographic keys. Clearly, current technology and best-practices cannot keep up with the rate of spread or the efficacy of modern-day attacks.

A new approach must be adopted that can effectively protect users' identities and their sensitive data. This solution cannot depend on the integrity of the host PC which is notoriously insecure, often unremediated and clearly the most significant vector for the spread of wide-scale attacks.

This paper presents ten requirements that, if complied with properly, effectively protect against some of the most common and devastating forms of cyber attack. First these requirements are explained, followed by a discussion of how these address the following types of attack families:

i. Spoofing
ii. I/O Sniffers
iii. Stack / Runtime Analysis

Biogy's *Confidentiality Identity and Authentication Protocol*, a novel approach to protecting individuals, their online identities, data and transactions is the first implementation of these requirements.

BACKGROUND

A system's security assurance capabilities must be considered not only on the merits of its independent properties, such as cryptographic methods and protocols, but also in terms of its operational structure. Much of the state of the art security practices follow standards and definitions that are not understood systemically and often times, system implementations leave large, unattended gaps and weaknesses. An example of this is the common implementation of online authentication systems that "leak" authorization information by providing different response behavior (content and/or timing) to bad username or password inputs -- thus opening the door to side-channel attacks.

Beyond treating security assurance from a complete systems perspective, data must also be understood in terms of its security value and protective resources allocated accordingly. For example, there is a significant difference between data contained in a communication channel containing unencrypted web traffic from a public news site, versus a channel carrying encrypted web content from a user's online banking site. A comprehensive solution must facilitate switching between sensitive and non-sensitive data contexts, while maintaining an impenetrable boundary between the two.

A NEW APPROACH

The ten requirements presented here represent a systematic approach for treating all sensitive data exchanges as secure transactions and also presents the necessary system properties to secure these. While contemporary operating systems and applications will continue to be susceptible to the ever-evolving slew of cyber attacks, a system following these requirements can still effectively protect a user's identity, their selected data, and their transactions -- the most crucial of informational exchanges -- so that if a system is compromised, damage will be contained and a user's credentials or data will not be accessible for leverage in subsequent attacks.

A secure transaction is defined as an atomic operation that is assured of availability, confidentiality, integrity, authenticity, authority and accountability. To this end, the following requirements must be met by any secure transactional system operating across an insecure Internet:

I. Any operation that must run in a trusted environment shall be treated as a secure transaction.
II. A secure transactional system shall assume that the Host and Network domains are untrusted.
III. A new notion of a *User Domain* shall be implemented as a secure module with a *minimal degree of programmability*.
IV. The *User Domain* shall be maximally interoperable with the Host and Network Domains.

V. A *Security Operational Stack* shall be maintained and information from a higher layer shall never leak to a lower one.

VI. Authentication operations shall be decentralized, while authorization operations shall be centralized.

VII. Every authorization operation shall be coupled with an authentication operation.

VIII. No pins or passwords shall be stored anywhere on the system.

IX. Multi-modal biometric authentication shall be seamlessly integrated with cryptographic operations.

X. Static tokens shall only be entered into the secure user module, while dynamic tokens shall be used for all necessary external operations.

I: Any operation that must run in a trusted environment shall be treated as a *secure transaction*.

The field of computer science defines a transaction as an individual or indivisible set of operations that must succeed or fail *atomically* (i.e. as a complete unit that cannot remain in an intermediate state). Further, a successful transaction alters a system from one known, good state to another, while a failed transaction does not (save for logging). To achieve this stateful functionality -- particularly in systems that handle concurrent transactions -- rollback, rollforward and deadlock handling mechanisms must be employed to assure atomicity and system state integrity.

The notion of a secure transactional system must assure the following properties:

i. *Availability:* Having timely and reliable access to a transactional resource.
ii. *Confidentiality:* Ensuring that transactional information is accessible only to those authorized to use it.
iii. *Integrity:* Ensuring that transactional information is protected from unauthorized modification.
iv. *Authentication:* Ensuring that transactional resources and users accessing these are correctly labeled (identified).
v. *Authorization:* Ensuring users access rights to transactional resources.
vi. *Accounting:* Ensuring that a transaction cannot be repudiated.

Any operation that handles or provides access to data deemed too sensitive for an untrusted environment (i.e. any private data) must be treated as a secure transaction to ensure that information leakage does not occur.

II: A secure transactional system shall assume that the Host and Network domains are untrusted.

A computational device is defined as a collection of hardware, and in some cases additional firmware and/or software, that receives digital or analog inputs through a user or network interface, processes logical instructions on the inputs (in a self-contained manner) and outputs the results of the operations though another user interface – for example, a computer screen. In some cases, computational devices are communicatively-coupled to other computational devices. In some cases, a communicatively-coupled, computational device contains logic, stored in memory, to perform the functions of the *Host Domain*, which is defined next. The host domain refers to the computing environment executing in the personal computer, mobile phone or other device that the user physically interacts with to access resources. *Physically interacts* means the user may type in a password into the keyboard of the "host domain" computer or even place his or her finger on a sensor connected directly to the "host domain" computer. The host domain renders the user interface to the secure application being accessed. In other cases, the host domain may refer to a mobile phone containing a processor that is executing an operating system that supports a web browser.

A computational device may perform *routing operations*, passing signals between other computational devices. The *Network Domain* refers to the communication infrastructure -- comprised of computational devices, as well as physical and virtual media -- that enables the transmission and routing of signals between distinct computational devices. Sometimes the network domain may refer to the interconnected, Internet Protocol (IP) networks that comprise the Internet. In other cases, the network domain may refer to a bluetooth connection between a user's mobile phone and their wireless headset. The network domain is comprised of host domains and network communications between these host domains. Figure 1 shows some computational devices acting as host domain devices and others acting as network domain devices.

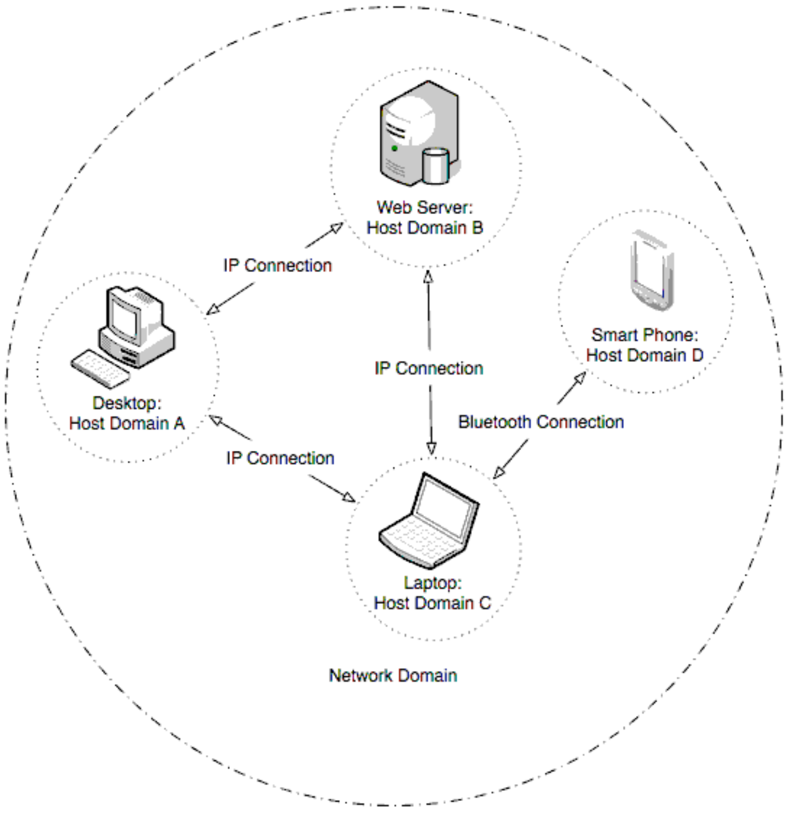

Figure 1

Historically, in the design of secrecy systems, there has always been a distinction between two domains -- trusted and untrusted. The trusted domain processes and encrypts information for subsequent transmission through an untrusted medium or channel.[3] The objective of secrecy systems is to maintain a well-defined impermeable boundary between these two domains. With the invention of the Internet this boundary has become unclear and security breaches resulting in data loss and compromise have increased dramatically over time.

The Internet from its inception was optimized for intercommunication and interoperability, and security assurance is still largely seen as a problem at the application layer (in the OSI model). All modern-day operating systems contain some mechanism(s) to distinguish between trusted and untrusted domains, which in the Internet model are the host and network domains, respectively. Implementations of this mechanism are most often protected by the use of static tokens and are host-based, which assume that the host is trusted (i.e. entering a fob-generated OTP into an unverified browser).

Computer worms and an ever-increasing amount of operating system and application vulnerabilities all too often result in successful breaches of this critical boundary and significant subsequent data and operational losses. For this reason, an effective transactional assurance mechanism must assume that the host domain is untrusted, along with the network domain.

III: A new notion of a *User Domain* shall be implemented as a secure module with a *minimal degree of programmability*.

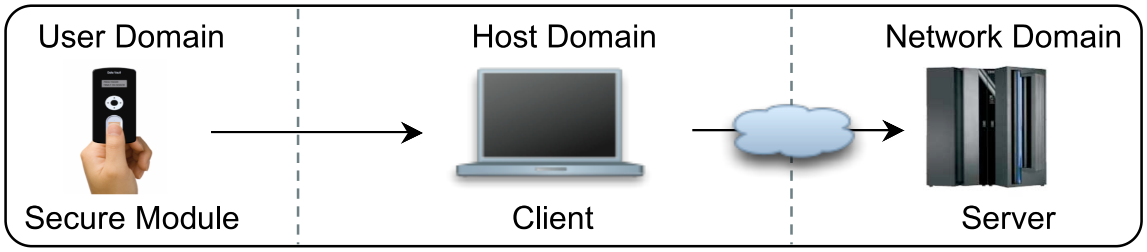

Figure 2

The user domain is functionally distinct from the host domain and the network domain. The user domain is implemented as a *secure module,* an operationally-independent, communicatively-coupled, restricted computing environment with the following properties and mechanisms:

i. A minimal degree of programmability.
ii. Minimally-accessible and error checked communication exchanges.
iii. All communication exchanges are executed as secure transactions.
iv. Static authentication factors are entered only into the secure module interface.
v. No static authentication factors ever leave the secure module, even if encrypted.
vi. Physical tampering or breaching of secure module will wipe all plaintext secret and private keys, any other critical security parameters (CSP's) and operational instruction set -- i.e. compliant with FIPS 140-2 Level 3[4].

The notion of a *minimal degree of programmability* can be further defined as a computing environment with these two properties:

i. A finite, immutable, operationally-restricted instruction set -- i.e. no OS or application that can be updated.
ii. An encrypted and restricted data set.

IV: The *User Domain* shall be maximally interoperable with the existing Host and Network Domains.

With the user domain being implemented as a secure module, maximum interoperability with the host and network domain is essential. The host and network domains represent a very large variety of network infrastructure, operating systems and applications. For this reason, the communication coupling must be based on cross-platform standards. This includes USB, Bluetooth and Near Field Communications (NFC). Also, the secure module should integrate, where it is deemed appropriate, with existing secure solutions such as VPN's and PKI systems.

Additionally, all secure module operations should be self-contained and any application on the host domain only routes encrypted transactions transmitted from the secure module.

V: A *Security Operational Stack* shall be maintained and information from a higher layer shall never leak to a lower one.

A securely-implemented transactional system must fulfill the following functions:

i. *Availability*
ii. *Confidentiality*
iii. *Integrity*
iv. *Authentication*
v. *Authorization*
vi. *Accounting*

The operational stack order represents that *Availability* has a higher priority over all other functionalities in the stack. As an example, a denial of service attack would knock out Availability so that all other functions would become irrelevant in accessing that resource. *Confidentiality* has the second highest priority. The operational stack is complementary and backward compatible with current implementations of these functions. The confidentiality, integrity and authentication stack operations are fully compatible with NSA Suite B cryptography.

For *Confidentiality*, key exchange can be implemented with Elliptic Curve Diffie-Hellman[5] and encryption can be implemented with AES-256 , FIPS 197[6]

For *Authentication*, the Elliptic Curve Digital Signature Algorithm, FIPS 186-2[7] can be used.

For *Integrity*, hashing can be implemented with Secure Hash Algorithm - FIPS 180-2, using SHA-512[8]

VI: Authentication operations shall be decentralized, while authorization operations shall be centralized.

Compromising one node should never compromise the rest of the system. In current biometric systems, the biometric data or prints are stored on the backend, typically in a database. In this scenario, a backend compromise would result in massive identity theft. Additionally, since biometrics are immutable, any significant breach -- say a database with biometric data for a million users -- would essentially make biometrics useless as an authentication factor for those users.

Therefore, biometric data should not be stored on a backend server or database. User credentials should be decentralized on a secure module. Only dynamic tokens should be sent to the backend server. Thus, a break-in of the system would not compromise any biometric data and helps protect a user's identity. In such an implementation, backend break-ins would only compromise user identifiers and big numbers. With a breach of this information, the system can be easily reset.

Authorization, on the other hand, should be centralized so that user access to system resources can be administered from an aggregate control point. For example, locking out a disgruntled employee is a procedure that would not involve the user and would be carried out by an authorized system administrator.

VII: Every authorization operation shall be seamlessly coupled with an authentication operation.

There are two main authorization system paradigms that are utilized in contemporary transactional systems. There is the Access Control List (ACL) approach and the capability-based authorization approach.

Most common is the ACL approach, which is implemented as a list with permission / user relationships, with which accessor processes verify that actions being carried out by the process owner are permitted. For example, in the case of the Unix file system (UFS), the “permissions list” is implemented as numbered inodes containing file metadata including ownership and permissions. A common problem with ACL implementations is that the list only contains a reference to the a user’s identity which results in common, well-known vulnerabilities (i.e. Privilege Escalation, Confused Deputy, etc).

Capability-based authorization systems require that the accessor process possess a token of authority (i.e. the capability) in order to carry out a requested action on a particular resource. Although these systems offer higher level of assurance than ACL-based systems, they depend on capability sharing amongst users, which essentially amounts to shared tokens -- an approach susceptible to all static token attacks (i.e. Sniffing, Spoofing, etc).

With the introduction of the user domain, secure dynamic tokens can be used to authorize resource rights, while simultaneously authenticating a user’s identity with every authorization operation.

VIII: No pins or passwords shall be stored anywhere on the system.

This prevents PIN-hacking attacks and cumbersome cryptographic protocols and methods needed to secure the PINs and passwords on the system. In addition, this method eliminates protection and maintenance of passwords and PINs on the system.

In addition to keeping PINs and passwords off the untrusted host and network domains, an ideal system would use mathematical techniques so that cryptographic keys, passwords and PINs are not stored on the secure user module. If a foreign adversary with sophisticated technical resources were to capture the secure module in the field, they would not be able to read or reconstruct the keys, PINs or passwords even with advanced methods such as UV reading of memory. PIN and password security should also be resistant to reverse engineering attacks.

IX: Multi-modal biometric authentication shall be seamlessly integrated with cryptographic operations.

In standard identity management systems, there is an on/off authentication switch between a biometric authentication and the release of cryptographic keys in applications using cryptographic protocols. This leaves a vulnerability (seam) between the biometric authentication and the cryptographic operations.

As an alternative, an ideal system should perform all cryptographic operations on a secure module where the biometric authentication occurs. The lack of an operating system on the secure module reduces the degrees of programmability, greatly diminishing security breaches and attacks between the biometric authentication and the cryptographic operations required in the user domain.

X: Static tokens shall only be entered into the secure user module, while dynamic tokens shall be used for all necessary external operations.

Static tokens are PINs, passwords, biometric data and other user information. They are user friendly because they are static but they are also susceptible to replay attacks. For this reason, they must entered directly into a secure user module. This prevents key logging and other types of malware capturing these static tokens.

Dynamic tokens prevent key logging, sniffing, resubmission attacks, and replay attacks. The dynamic tokens must be implemented with non-autonomous systems and NSA-approved cryptography. This greatly increases the entropy and Kolmogorov complexity (unpredictability) of an ideal system. Advanced methods in dynamical systems can be used to predict the behavior of complex systems such as cryptographic algorithms and other important algorithms in a security system. These methods have been shown to build highly effective predictive models of complex systems when standard statistical methods only indicate that the behavior of the complex system is coming from an assumed Gaussian source[9]. The generation of dynamic tokens must be designed with these advanced and possibly unforeseen attacks in mind.

Addressing Attacks

In this section we look at the efficacy of the ten requirements relative to three common types of attack families: Spoofing, I/O Sniffers and Runtime Analysis.

Attack Family: Spoofing

Common Forms: Phishing, DNS cache poisoning, man-in-the-middle and cross-site scripting (XSS).

Description: Spoofing attacks take advantage of surmountable authentication mechanisms, where a malicious user, process or resource masquerades as a valid one, thereby gaining an illegitimate advantage. In many of these attacks the primary objective is to obtain user credentials that can be later used for unauthorized activity. In the case of cross-site scripting attacks, it was estimated that nearly 70% of websites in 2007 were open to these attacks[10]. In terms of monetary loss, also in that same year phishing attacks affected 3.6 million adults, at a loss of US $3.2 billion[11].

Countermeasure: There are two primary mechanisms by which spoofing attacks are addressed by the ten requirements. First, the secure module stores the necessary PKI certificates, OTP seeds and any other user credentials that should not be stored on the host PC (where they could be altered). This approach can leverage existing authentication mechanisms, but in a far more secure configuration -- providing a trusted mechanism for essential two-way authentication. Also, particularly for phishing attacks, using a secure module that accepts a fingerprint and can verify for the user the authenticity of a secure site, removes the human element of having to assess whether a site is a legitimate place to enter user credentials.

The second mechanism is the use of one-time passcodes, for every authorization operation, which as mentioned above would also be seamlessly coupled with an authentication operation as well. This approach would secure against common cross-site scripting attacks that leverage previously gathered user data (i.e. a web browser cookie).

Attack Family: Input / Output (I/O) Sniffers

Common Forms: Network sniffers (i.e. tcpdump, ethereal), keystroke loggers and RF capture

Description: Security attacks categorized as I/O sniffers present a significant risk because they aim to capture input and output streams from host PC processes that contain sensitive data to be used in latter attacks. This includes passwords, PINs, biometric prints or templates and any other user data that must remain private.

Countermeasure: Since all static authentication tokens are entered directly into the secure module interface, this prevents key stroke loggers and other malware from capturing these authentication tokens. Additionally, the minimal degree of programmability that prevents new code from executing in the secure module assures that I/O sniffers can not be installed.

Moreover, because one-time passcode authentication factors may only be used once, these dynamic tokens thwart the I/O sniffers executing in the untrusted host and network domains. Catch and resubmit attacks that may be added to an I/O sniffer are thwarted because only the secure module has access to its private certificate, which signs the transaction along with the one-time passcode.

Attack Family: Runtime Analysis

Common Forms: debuggers (i.e. gdb), disassemblers (i.e. IDA Pro, OllyDgb) and emulators

Description: Runtime analysis attacks are perhaps some of the most sophisticated attacks, which are comprised of observing a running process and capturing the executing instruction set and data contained within through the use of specialized software. These attacks are very difficult to prevent if an attacker has access to an operating environment which is running a critical process that may contain or route critical data such as keys, certificates or any other sensitive information.

Countermeasure: Since critical transactional functions run on a secure module with a minimal degree of programmability, runtime analysis cannot occur in the secure module environment. Additionally, tampering with the physical device to extract the instruction set for execution on an emulator results in destruction of the data set.

REFERENCES

1 http://online.wsj.com/article/SB123914805204099085.html

2 Joseph Menn. Fatal System Error. PublicAffairs, New York, 2010. ISBN 978-1-58648-748-5.

3 http://www.aemea.org/crypto/shannon1949.pdf

4 http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

5 http://csrc.nist.gov/groups/ST/toolkit/documents/SP800-56Arev1_3-8-07.pdf

6 http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

7 http://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf

8 http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf

9 http://www.edge.org/q2006/q06_11.html#kosko

10 http://www.csoonline.com/article/221113/Software_Vulnerability_Disclosure_The_Chilling_Effect

11 http://www.gartner.com/it/page.jsp?id=565125