WHEN THE SOFTWARE GOES BEYOND ITS REQUIREMENTS
-- A SOFTWARE SECURITY PERSPECTIVE

Jing-Chiou Liou
Department of Computer science
Kean University
Union, NJ 07083, USA
jliou@kean.edu

Abstract—Evidences from current events have shown that, in addition to virus and hacker attacks, many software systems have been embedded with “agents” that pose security threats such as allowing someone to “invade” into computers with such software installed. This will eventually grow into a more serious problem when Cluster and Cloud Computing becomes popular. As this is an area that few have been exploring, we discuss in this paper the issue of software security breaches resulting from embedded sleeping agents. We also investigate some patterns of embedded sleeping agents utilized in software industry. In addition, we review these patterns and propose a security model that identifies different scenarios. This security model will provide a foundation for further study on how to detect and prevent such patterns from becoming security breaches.

Keywords—Software security, security code, software security model, software agents, software assurance.

I. INTRODUCTION

In modern society, the Information Technology (IT) has played a significant role among people’s daily life. Almost every modern electronic device, instrument, or system is a digital system. Each digital system is pre-installed a software to provide a Graphic User Interface (GUI) for operation and to control the functionalities of the system.

For personal devices such as cellar phones and GPSs, device drivers are the real driving power of such modern gadgets. On the other hand, many grand scale systems ranging from aircraft flight control, network and power plant monitoring, to manufacturing pipeline, etc., rely on software to perform their functions. In addition, many organizations utilize Information Systems (IS) to process and analyze data, to assist in decision making, and to serve as a data repository for records keeping. Not to mention that the Internet requires software supports to enhance its usability and the Electronic-Commerce cannot function without the web technology.

All these drivers, GUIs, data process engines and databases in those digital systems are actually software. Therefore, we may say that all of these modern technologies are built upon a foundation of software applications. Without the software installed in the digital systems, those digital systems are just pieces of steel and wires.

Due to the fact that all of these digital systems require software, software development has become a major industry today. With the demand of software in many digital applications, the needs for professional software engineers grow exponentially in the past decade. Many institutes, including colleges and technical schools, have trained tens of thousands, if not millions, IT professionals worldwide. However, the supply of IT professional is far less than the demand. To meet the demand, software industry turns to other options such as outsourcing, off-shoring, and even hiring people that is not fully trained professionals.

The trend of utilizing such options in software industry has impacted not only on software quality, but also on security of the software developed. We will discuss such impacts and issues of software security breach in the following section. We will identify some patterns of embedded sleeping agents that may cause such security breach in Section 3. We then review the current methodologies for software review and inspection utilized in software industry to find an efficient way for manually catching such agents. Finally, in Section 4, we will develop a security model based on the patterns identified in the Section 3.

II. ISSUES OF SOFTWARE SECURITY BREACH

A. Threats

For the last decade, software security has been getting more and more attentions as the information technology becoming the necessity of daily life. Viruses and hackers hacking into some critical IT systems are often seen in the news. Governments, as well as commercial industries, spend significant efforts to prevent information security breach from invasion and from virus spreading.

Software security breaches are threats that exploit vulnerabilities of software system to damage assets of an organization. These threats can be categorized by the resulted damages. One model, STRIDE [1], lists the following categories:

- Spoofing: The attacker uses fake or somebody’s identity.
• Tampering: Data validation of user data does not occur.
• Repudiation: System is not capable to trace improper request.
• Information disclosure: Confidential information leaks out intentionally or unintentionally.
• Denial of service (DoS): DoS attacks make servers temporarily unavailable to their valid users.
• Elevation of privilege: A valid user gains more privilege on a system.

Security is meant for the protection of organization assets. Security measures can be defined as: Prevention, detection, and reaction. As security software such as anti-virus program becoming one major sector in software industry, there is little awareness on the rising of the software security issue caused by software embedded agents. As a matter of fact, according to a report [2], the security of software is threatened at various points throughout its life cycle, both by inadvertent and intentional choices and actions taken by “insiders”—individuals closely affiliated with the organization that is producing, deploying, operating, or maintaining the software, and thus trusted by that organization—and by “outsiders” who have no affiliation with the organization.

Moreover, statistics show that attacks from insiders account for a majority of incidents and the largest proportion of damages [3]. With current trend of software development outsource and off-shore, many evidences from current events have shown that, in addition to virus and hacker attacks, more software systems have been embedded with “agents” that pose security threats such as allowing someone to “invade” into computer installed with such software.

B. Embedded Sleeping Agents

One example is that, in February 2008, a virus was discovered on a digital photo frame with its roots in China [4]. The virus, or to be exactly Trojan Horse, gathers passwords for online games. While this does not sound very harmful, it may have tremendous effects on the industry. The virus, called Mocmex, is said to Identify and block most antivirus software packages and firewalls (including the Windows built-in firewall). Mocmex downloads data and files from remote locations. Furthermore, Mocmex is able to enter computers or portable storage products such as USB sticks.

Another example is that INTEL recently announced the development of Remote Wake technology into its chip sets that enables consumers to "wake up" their home PC from energy-efficient sleep mode to access all their media, anytime, from any internet-connected device. In addition, a study [5] by researcher and technical writer Kris Kaspersky shows “Intel CPUs have exploitable bugs which are vulnerable to both local and remote attacks which works against any OS regardless of the patches applied or the applications which are running.” Examples also include Microsoft’s RPC [6], malware [7], and even security problem in antivirus software [8]. More recently, Dell's delivery of motherboards with malware [9] is the latest warning of such the security threat.

To attack on the security breach, firstly, we need to systematically study the cases (patterns) and develop a model for the problem. This will require collaboration with software industry to collect possible cases. Secondly, the cases and the model will then be analyzed for any effective and efficient remedies. As this is little known to the researchers in the field and few have been eyed on the issue, the remedies at the early stage can be on the process side. In other words, we can investigate on the software development process and discover any preventive action to be implemented into process activities.

The last step will focus on a development of tools for agent detection automation. These tools may be embedded with any software development process and can help software testers to uncover any sleeping agents that are difficult to be detected by the normal testing activity. Implementation of both remedies of manual process activities and automation tools needs collaboration with software industry.

The study presented in this paper focuses primary one the first phase that identifies patterns and constructs a security model. The study also covers partially discussion on software process that shall be able to manually uncover any such patterns to prevent security breach from happening.

III. PATTERNS OF EMBEDDED SLEEPING AGENTS

To be considered secure, software must exhibit three properties: Dependability, trustworthiness, survivability. The objective of secure software development is to design, implement, configure, and sustain software systems in which security is a necessary property from the beginning of the system’s life cycle (i.e., needs and requirements definition) to its end (retirement) [2].

In many software developments, additional blocks of codes are implemented in the software applications beyond the systems’ functional requirements. Most of the time, the implementing of such “extra” codes is necessary for the software application. Their purposes may vary, but their intentions are not harmful to anyone.

However, in some cases, blocks of codes can be added beyond the requirements for personal purposes. In this section, we will show some of the commonly seen patterns of adding blocks of code for very unique intention that can become a security concerns.

To be clear for the discussion, we firstly define an agent as a block of code that is inserted into the software without a user requirement. By the definition, not all agents should be considered as a security threat. In many cases, agents make active most of the time to perform some desire functions, although not appreciated by the users. Some agents may be activated less frequently, but serve their purposes of performing functions required by the system.

An agent can be a security threat only if the software developer intends to do so. Some of the patterns presented in
the following sub-sections are actually used currently in software development for legitimate purpose. But events show that they can be also used for other purposes that become a security threat.

A. Software-Computer Binding

Software agent uses some specific hardware system information to identify the computer that the software is installed. Such hardware information could be the IP address, MAC address or other addressing identifiers of the computer. Once the software registers the computer identifier, the software will not function correctly if installed on computer that has different identifier.

The primary purpose of this commonly used method is to prevent the user from cloning the software or installing it onto other computers without legal licenses. Software applications are usually not free, and not cheap, too. Some users may obtain one legal license and try to install onto many computers. Although it is illegal, laws cannot stop people from doing so. Therefore, some software application has built into the software an agent for software-computer binding. Software can function perfectly only on the computer whose identifier is registered.

However, the same technique can be applied to acquire computer configuration on other computer for dreadful use. For example, the computer where legitimate software is installed can be shut down periodically for maintenance. A ghost computer installed with the software and connected to the same network can be activated during the time and function like a legitimate system to process information such as business transactions or other restricted data. Before it is detected, the ghost computer can disappear already and leave no trace.

Regular IT process will not do so because it will produce two separate set of system data or databases that need to be merged later. But if the second database is intended for a security breach, it poses a threat to any systems that have confidential data fed into them.

B. Back Doors

Create a back door to a software system is another common phenomena in software development. Its purpose is to allow software develop team to be able to look into production problems immediately when they do happen. Formal software process requires software development team to grant access to production from customer. This action may be time consuming because business cannot be interrupted and business information is mostly restricted.

However, not every production defect can be reproduced in the development environment, and the pressure to fix the problem in timely fashion will force the development team to create a back door to the software production system. It sometimes also provides a good excuse for the development team to attribute a production problem as an operation error if it can be secretly fixed before the problem is escalated.

This back door is different from the “Trap Backdoor” frequently discussed in information security. In trap backdoor, network intrusion uses port scan to find network host open ports. INTEL’s Remote Wake technology and Microsoft’s RPC can be considered as different types of backdoor techniques.

There are two types of back door that can be seen today. Nonetheless, depending on the software architecture design, each type of back door will serve its purpose as a security breach.

1) Special Account login

In this pattern, the software development team will use a unique client account login and password specially created for the purpose to access to the production system. This login can be a super user or an administrator account. But sometime it can be just another common user account that is not used frequently.

When a production defect happens, development team can log on to the production system either to check on any log files, or to just try to reproduce the error. However, due to the attribute of special user account used by this pattern, it is quite easier to leave trace of account access information on the production system. A careful production system administrator can discover such access and trace the access back to the development team. Therefore, this pattern is considered less technical challenge.

2) Multiple instances of the same Account login

Another pattern in creating a back door is to allow multiple instances of the same common user account that is used daily. Since the password can be changed periodically by the current user, to utilize the pattern, the specific account is actually accepting two passwords: one is set by the current user and another is only known to the development team.

The concept of this pattern is very simple. For many users of software application, they could log on to the application either from a desktop, a laptop, a PDA, a smart phone, or even all of them at the same time. In addition, many web applications, such as a web email or an ERP system, allow multiple instances, disregarding if they are from the same computer or different computers.

Comparing with the special account login discussed in previous subsection, development team accessing to the production system using this pattern is not easily to be distinguished from the regular system activity, especially if the production access by the development team is short and infrequently.

C. Event-Based Release

One frequently used method of stopping user from using software that exceeds its trial or licensed period is to set a time stamp or counter to the software. A due date or an expiration date is pre-set in the software. When the software reaches the limit set by the time stamp, the software performance degradation commences or a temporary back door is created for developer accessing to the system. The method is now applied beyond its original purpose to create a security breach.
In normal use of this technique, a warning message will pop out regularly to remind users to either purchase a new or to renew the license. Some applications will terminate the operation after certain probation period; others will allow the users to use the degraded function continually.

If the technique is used for the purpose of security breach, the agent will activate some specific functions that allow the attacker to gain control of the software system. In the case, it is usually referred as Logic Bomb. These scenarios could include creating a back door, stop certain functions, and/or add some functions to the software.

For example, an agent could create an account to allow access to the application on February 29, which can happen only once every 4 years. As this is a case that is so rare, almost no one will pay attention to such access. An agent could also stop security defense system, such as a firewall, or to expose a port for a short period of time to allow hackers to attack the application. An agent could even activate some function on certain day to collect system data, such as users/customer information, for a short period of time and shut down such function immediately after the timer reaches the limit.

The event-based release technique is a powerful technique for software security breach. It is quite possible that this technique might leave no, or at least minimum, trace of the action. However, it should be noted that the technique can be implemented to work with other patterns to produce enhanced actions in security breaches.

D. Program Checksum

Similar to the watermarking used for document and image files, software developers use some techniques to check if the software has been modified by unauthorized person. One simple technique is to implement a program checksum. Validation of program checksum is performed periodically or every time when the application is initiated.

The common use of program checksum is to verify the origin formation of the software and the warranty of the software license. If the program checksum is found to be different from its original status, the developer may assume the software has been unauthorized altered, and hence, the warranty is voided. However, that finding may rely on network connection. If the firewall prevents that notification from happening, the developer cannot use the technique efficiently.

To resolve that network connection issue, some developers insert an agent into the application to work offline. In other words, the agent performs program checksum without communicating with the developer. When the agent finds the program checksum is varied from its origin, it can advise the user to contact the developer for such issue or even stop the application from functioning.

From software security point of view, an agent can be inserted to change the program checksum. This can be done directly or indirectly. To do it indirectly, the agent can add into the software or deleting from the software some code that is pre-compiled and interfaced with the application. The change of program checksum will result in malfunction of the application.

This technique is a perfect partner of the time release pattern. An agent has been set to activate some function at certain point of time. Since there is no intrusion can be traced, it is hard to tell what happened to the software.

Both the software developer and the user have a good reason to believe that it is the fault of the other. Nonetheless, it is a fact that the software has been modified. And the malfunction of the application, no matter how short it is, has caused the damage to the business.

IV. A SOFTWARE SECURITY MODEL

It is not uncommon that software developers intentionally create agents mentioned in the Section 3 in a software application for some purpose (for example, test the strength of the application), but unintentionally leave the agent in the software (after completing its unit test.) Software testers have no way to identify such agents as those functions are not in the system requirements.

There are some existing high level security models associated with the software life cycle. An example is presented by Microsoft™ [10]. The concept behind it is derived from the software testing or quality assurance integrated in the software development life cycle (SDLC). The term “software assurance” that comprises software reliability, quality, and security engineering is a new subject for software development. A model [2] uses such concept to present necessary security assurance activities throughout the phases of a SDLC.

However, very little, if not none, study had been presented for software security model that includes both the external attacks on the software vulnerabilities and the internal threats
resulting from the agents. In this section, we propose a model that fills this gap by categorizing the scenarios of security breaches from both external attacks and internal agents.

We distinguish an agent from vulnerability in our model. An agent is a block of code built intentionally or unintentionally into production software that carries a specific function known or unknown to the users of the application. Vulnerability is a weakness or defect of the software that is vulnerable to the security breach.

Figure 1 depicts a software security model that consists of actors, actions, and stages for actors to access and actions to happen. An actor is a person or device that can access to a software application through connection crossing over the boundary of an enterprise system(s). A security action is an event or activity initiated by an agent or caused by an attack to a vulnerability built into the software.

From Figure 1, we can categorize the scenarios of security breach into 5 cases:

A. External Attacks

These are the most commonly referred software security threats. In this scenario, an actor initiates an attack to a vulnerability of a software application through network connection. Hence, it is a type of online security breach.

Due to the fact that the existence of vulnerabilities in today’s software is guaranteed, external attacks are easy jobs for actors (hackers or anyone who are interested in finding such vulnerabilities.) Therefore, many attentions have been drawn into this scenario. Security software, such as firewall and anti-virus program, are developed for detecting and blocking intrusion. And security software has become a major sector of today’s software industry.

B. Hostile System Actions

Instead of attacking by an external actor, software vulnerability could be attacked internally by an agent embedded in the software. This is usually not what people have learned from the public news.

When such attacks happen, they are most likely being classified as attacks caused by adware or spamware. However, adware and spamware are usually an extension part of software that is downloaded from the Internet.

Comparing with such situation, hostile system actions are caused by agent(s) embedded in the software that is legally licensed and believed to be trustworthy. These agents attack on system vulnerabilities from inside and probably will not be detected by any security software like firewall or antivirus or security function built in the software being attacked. This scenario is an offline type of action, meaning it does not require a network connection.

C. Unauthorized Communications

A hostile system action attack on software vulnerability and may cause software malfunctioning. On the other hand, an agent may create an unauthorized network connection no listed in the requirement to commence a communication between the software application and an actor.

As the purpose of the communication may vary, an agent could transmit some confidential information beyond system boundary. For example, an agent may collect system data, such as personal information or financial data, and deliver it to someone for criminal activity.

This action requires a network connection; hence, it is an online action. This action could possibly be detected if the software behavior is closely watched. And the actor is also could be easily traced.

D. Functions Blocking

Similar to the internal attack of a hostile system action, an agent could attack not on software vulnerability, but a software function. This action is offline which does not require a network connection. It acts from inside of the software and works on to degrade or even to disable certain software function(s).

Since there is no communication involved in the action, it is difficult to be identified; even if the software behavior is being monitored. Also, because there is no actor interfacing with the software system, it will not be considered as an external attack. Eventually, this security breach will be treated as a software defect.

If the action blocks some core function completely, it may bring down the software, or even the whole enterprise system. Because the security breach is random, and probably not repeatable by any test scenario, the agent could pass security test during the process of fixing the “defect.”

E. Unauthorized Interfaces

In this type of action, an agent resided within a software establishes an interface that is not defined in the system requirements to across the software boundaries. This is an online action but involves with no actor. However, it may be an offline action if both software interfaced are installed on the same computer. There are usually many software applications utilized in an enterprise system, such as an ERP suit. Interfaces exist between software applications to share some common information. But confidential information may be processed solely on certain software and is not shared with other applications.

A security breach could result from an authorized interface between two software applications established by an agent. Confidential information held by a software can be transmitted through such interface into another software with agents embedded into it. In addition to the agent that establishes the interface, some others could in turn transmit that confidential information across the system boundary out to some actor.

The interface could be temperately established and the period of time such interface exists could be also very short. If the action is offline, meaning it does not require network connection, the interface can be difficult to be detected. But it is quite possibly to be reproduced if such action is detected.
V. CONCLUSION

Software is expected to be reliable, trustworthy, and free from defects. However, that may be not the case in reality. Both quality and reliability, but not the trustworthiness, are known to be major concerns to people who use software regularly. Yet, people install software with full trust for whatever have been implemented in the software.

Current events have shown that software usually comprises features that beyond users’ expectation. In addition, more and more cases show that software features go beyond its user requirements and may cause security breach. To address the issue that few have been explored, patterns of agent built in to software that may be considered as security threats are discussed.

Moreover, a security model derived from these patterns has been presented. Details of different types of actions and possible associated actor are also discussed. This model provides a foundation for further study on how to detect and prevent such patent from becoming security breaches.

REFERENCES


[9] Dell's delivery of motherboards with malware http://news.yahoo.com/s/pcworld/20100722/tc_pcworld/dellrevampshardwaretestinginwakenofmalwareissue;_ylt=ArPw995rYBIBqw7Hehf8U.iDzdAF_,_ylu=X3oDMTNxb0kxM3ByBGFzc2V0A3Bj29ybGQvMjAxMDA3MjlvZGVsbhJidmFtcHN0YXJkd2FyZXRle3Rphmdpbndha2VvZm1hbHdhcmVpc3N1ZQRwb3MDMTTeEcVJaA3huX3BhZ2luYXRlX3NlbWVhc2li6GlzdARzbGsDZGVsbhHJidmFtcHN0.