Jordanian Journal of Computers and Information Technology (JJCIT), Vol. 07, No. 03, September 2021.

INCREASING SECURITY IN MILITARY SELF-PROTECTED SOFTWARE

Carlos Gonzalez

(Received: 4-May-2021, Revised: 2-Jul.-2021, Accepted: 9-Jul.-2021)

ABSTRACT

The objective of this article is to describe a process methodology to increase security inside secure military self-protected software. Self-protected software is vulnerable to threats, most dependant on the software user. Therefore, detection by self-protected software of the current user is very important. The methodology includes three phases: detection of user, analysis of current state and reaction actions. The detection phase is comprised of assessing geographic location, time at present location and determining user kind (friend or foe). Analysis phase consists of analysing if self-protected software should be at present location, predicting future locations and assessing the location level of threat. Reaction phase includes determining immediate and delay actions if any and perform actions accordingly. Legal concerns are explained, countermeasures and covert actions are proposed and described. An analytical model shows that self-protected software that includes user detection provides more protection than self-protected software without user detection.

KEYWORDS

User detection, Secure military software, Secure self-protected software, Covert actions, International law.

1. INTRODUCTION

One important feature of military secure self-protected software systems is that these systems must be able to detect the current users of the system. This article describes a methodology and provides a list of useful guidelines to obtain user detection for military secure self-protected software systems, the analysis of the software current state and possible reactions to threats.

It is important to point out that this article does not include or discuss the use of covert electronic surveillance of any kind inside the self-protected software described in this paper [1].

The types of military secure software self-protected systems addressed in this paper range from the military software application running in a user laptop or cell phone, to the ones that generally are not connected to a network of any kind and/or are embedded software running on a specific device. Following is description previous research in areas related to this work.

1.1 Intrusion Detection Systems

Much work has been done on network and host intrusion detection systems to identify user-intruders in networks.

Amer et al. [2] provided researchers with a taxonomy and survey of current dataset composition and current Intrusion Detection System (IDS) capabilities and assets. These taxonomies and surveys aim to improve both the efficiency of IDSs and the creation of datasets to build the next-generation IDSs as well as to reflect networks threats more accurately in future datasets. To this end, this manuscript also provides a taxonomy and survey of network threats and associated tools.

Lunt [3] described the use of such tools for detecting computer system intrusion and describes further technologies that may be of use for intrusion detection in the future.

Zhang et al. [4] examined the vulnerabilities of wireless networks and argued for the inclusion of intrusion detection in the security architecture for mobile computing environments. They developed an architecture and evaluated a key mechanism in this architecture by anomaly detection for mobile *ad-hoc* network through simulation experiments.

Koch et al. [5] presented a model-driven approach to engineer self-protection for autonomous systems. The approach is integrated into model-driven security SecureUML for modeling access control and

C. Gonzalez is with the Facultad de Sistemas, Universidad Autonoma de Coahuila, Mexico. E-mail: gonzalezc757@gmail.com

supports the system designer in engineering self-protection rules to react to unexpected security vulnerabilities. Self-protection is specified by a set of transformation rules which restrict the security model. A graph-based semantics for the transformation rules allows to verify that security requirements are satisfied by the specified self-protection rules.

As the deployment of network-centric systems increases, network attacks are proportionally increasing in intensity as well as in complexity. Attack detection techniques can be broadly classified as being signature-based, classification-based or anomaly-based. Nashif et al. [6] presented a multi-level intrusion detection system (ML-IDS) that uses autonomic computing to automate the control and management of ML-IDS. This automation allows ML-IDS to detect network attacks and proactively protect against them. ML-IDS inspects and analyzes network traffic using three levels of granularities (traffic flow, packet header and payload) and employs an efficient fusion decision algorithm to improve the overall detection rate and minimize the occurrence of false alarms.

Elkhodary et al. [7] developed a methodology for designing security and designing security-aware adaptive systems that adapt in response to an attack. Adaptive application security requires all reconfiguration scales, automated detection and resolution of conflicts and the consideration of security features collectively.

Ankit Thakkar et al. [8] in their study reviewed the datasets developed in the field of Intrusion Detection Systems (IDSs). These datasets have been used for performance evaluation of ML- and DM-based IDSs. The study revealed that there is a need to update the underlying dataset to identify the recent attacks in the field of IDSs with improved performance.

Kyle Guercio [9] guide covered the industry’s leading Intrusion Detection and Prevention Systems (IDPS), along with a summary of key features to look for as you evaluate solutions.

1.2 Software and Hardware Specifically for Military

Much software and hardware work has been done specifically for military. Here are some examples of unclassified work.

Mahmoud et al. [10] reported information on network security in the aeronautical communication domain. Three fundamental research axes are explored. First, a quantitative network security risk assessment methodology is proposed. Their approach is based on the risk propagation within the network nodes. As study cases, the algorithm has been validated in the scope of the European industrial project entitled SESAR (Single European Sky ATM Research) and the Aerospace Valley FAST (Fibrelike Aircraft Satellite Communications).

Keller [11] reported that BAE Systems received a $4 million contract from the Navy for a quick-turnaround demonstration of a new radio frequency countermeasure system for the P-8A Poseidon aircraft. The electronic warfare (EW) system will be a lightweight pod mounted to the aircraft that will add self-protection capability to the Poseidon. It consists of a small jammer, a high-powered amplifier and the AN/ALE-55 Fiber-Optic Towed Decoy and lures enemy missiles away from the aircraft to the towed decoy.

Will developing intrusion detection capabilities meet the operational, performance and implementation goals of the US Air Force? To help ensure that they will, the MITRE C2 Protect Mission-Oriented Investigation and Experimentation (MOIE) project has been focusing on Air Force needs with a view to articulating them to commercial interests that may develop capabilities. This may help shape future funding decisions and may also provide a common framework for discussing issues.

LaPadula [12] explained a first cut at defining goals, capitalizing on customer and corporate experience with intrusion detection tools as well as knowledge of the problem domain. They created an information base about intrusion detection, providing a framework for discussing, refining and enhancing intrusion detection goals. One technological countermeasure is intrusion detection capability. Once detected, a variety of actions can be taken to thwart an attacker’s intentions. In the recent past, intrusion detection capabilities have been developed by both governmental and commercial interests.

Passive IDS includes such components as security cameras, motion detectors, thermal imaging systems, radar, card readers and keypad systems that are designed to monitor and limit entry to facilities. These systems are thought of as a kind of firewall used to keep out unwanted guests. Just like systems designed for online protection, IDSs have grown more automated over time with the addition of such things as voice recognition, facial recognition, license plate scanners and more.

Andone [13] explained reactive systems, also known as Intrusion Prevention Systems (IPSs), where the ability to respond in kind to suspicious activity is hardwired into the matrix. Like watch dogs, these systems not only observe, but also are tasked with providing feedback to system operators and security personnel.

1.3 Emerging Technologies

The methodology described below can be used for the creation of Runtime Application Self-Protection (RASP) applications, which represent an emerging security technology [14]. As described in Veracode [15] RASP resides on the server and embeds security into the running application. It then intercepts all calls to the system and ensures that it is secure by implanting validation of data requests directly into the application. RASP can be applied to Web and non-Web applications and does not affect the application design. Rather, the detection and protection features are added to the server on which an application runs. With RASP, enterprises are not building a secure application, but rather adding a "shield" to the code. If an application is defective, it will remain so even when protected by RASP. In other words, RASP is not going to make an app as secure as it would be if all the security requirements for the self-protected software were built into the app from start to finish. The methodology described below carries the advantage of the security built into the code from the start.

2. ASSUMPTIONS

For security reasons in this paper when we talk about an algorithm, procedure or methodology, we describe only the function's general purpose and how it is expected to work. We do not give details of the algorithms, procedures or methodology, nor how these should be implemented.

We also assume that for the military secure self-protected software system, including the algorithms and procedures described here, the only code available to the user will be the executable code.

We also assume that the design and implementation of the self-protected software include from inception in its specifications and requirements all the security needed for the self-protected software.

3. METHODOLOGY

It is important to clarify that the user detection function addressed in this methodology is not the detection of users coming through a network, but the actual user using the self-protected software.

The proposed methodology approach describes a process to increase security inside secure military self-protected software. Self-protected software is vulnerable to threats, most of which depend on the software user. Therefore, detection by self-protected software of the current user is very important. The methodology approach includes three phases: detection phase of user, where a process methodology is used to best determine physical location and the user kind (friend or foe) executing the self-protected software; the analysis phase, in which a rigorous analysis is done of the current user and state of the self-protected software; and a reaction phase where, given the current kind of threat taking place, a set of actions are performed. The complete architecture of the methodology process is presented in Figure 1.

3.1 Detection Phase

The detection phase is comprised of the following processes; assessing geographic location, time at present location and determining user kind. Figure 2 shows the flow diagram of the software processes comprising this phase.

3.1.1 Assessing Software Geographic Location

A key issue that self-protected software should be aware of is the current physical location of the device. To be a truly self-protected software, this software should be aware of its immediate surroundings/ environment. The environment will be different in friendly *vs.* enemy territory.

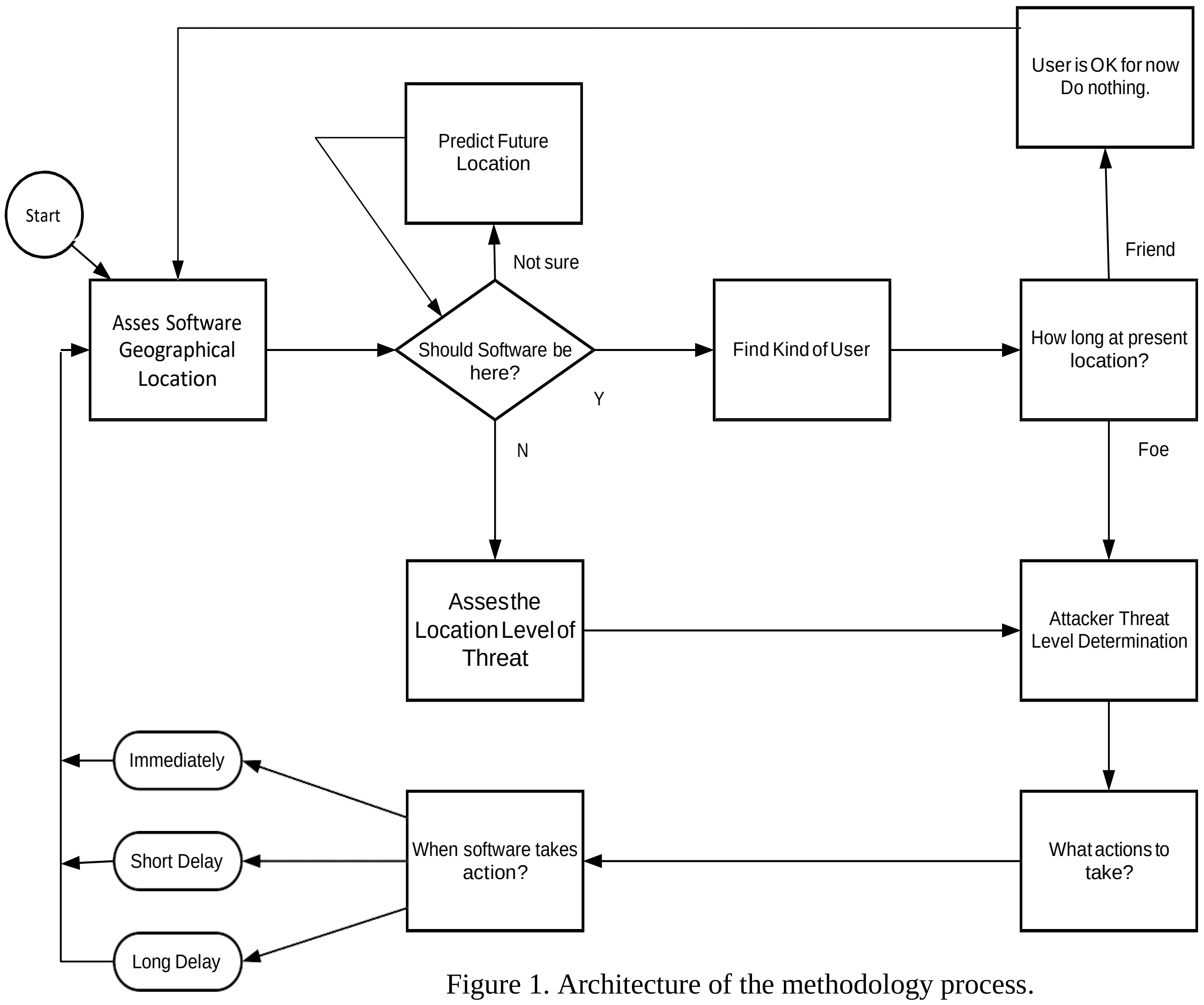
Start
Asses Software Geographical Location
Should Software be here?
Not sure
Predict Future Location
Y
N
Find Kind of User
How long at present location?
Friend
User is OK for now Do nothing.
Foe
Asses the Location Level of Threat
Attacker Threat Level Determination
What actions to take?
When software takes action?
Immediately
Short Delay
Long Delay

Figure 1. Architecture of the methodology process.

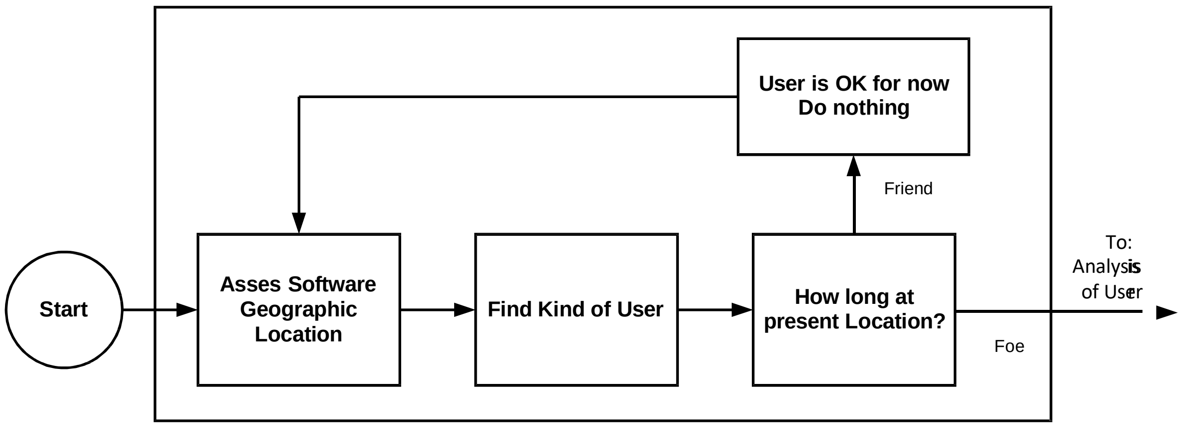
Start
Asses Software Geographic Location
Find Kind of User
How long at present Location?
User is OK for now Do nothing
Friend
Foe
To: Analysis of User

Figure 2. Detection phase.

The software self-protected actions and responses will vary according to the software current location.

Since the software's current location may change over time, one of the most important questions that one should ask is: Where is the software now? This basic question would not be easy to answer if the software does not have some sort of GPS.

If the original manufacturer is providing the hardware and the software, then the author of this article suggests that the following devices be included inside the hardware of a secure self-protected software system:

(1) At least two sets of GPS devices available to the software.

(2) At least one Wi-Fi mechanism to connect to the internet.

(3) At least one Bluetooth mechanism to receive and send signals.

(4) At least one mechanism to destroy our local hardware (See note below).

(5) An explosive to destroy an area in close proximity to our hardware (See note below).

(6) At least one heartbeat mechanism to monitor the devices installed in the hardware in order to ensure that all are working.

It is important to point out that item number 4: the mechanism to destroy the local hardware should exist only when the hardware is considered a specialized hardware (e.g. a sonar system, a tactical radio or a fire control system). It does not apply to cell phones or regular laptops. In such instances, when the hardware is not considered specialized, it would be better to have implemented a “disable” mechanism instead of a “destroy” one. Therefore, the software will have the “destroy” mechanism when running on an embedded system part of any military hardware. In order to avoid false-positives, it is recommended to have fail-safe mechanisms to ensure that when such drastic action is required; for example, communicate to the user when that what he/she is doing is illegal with consequences. If the warning (may be more than one) is not taken, then the destroy action will commence.

Also, on item number 5, an explosive to destroy an area in close proximity to our hardware having this option should be exercised only when dealing with highly sensitive software/hardware that is important to national security. The author firmly believes that in some cases, we have to react as if we were at war. In other words, we do not ever want to be in the hands of a foreign country that is our known foe. Again, as stated in item number 4, in order to avoid false-positives, it is recommended to have fail-safe mechanisms to make sure that such drastic actions are required. One such action could be a warning to the user, but if our intent is to continue in stealth mode, another action could be degrading the performance of the system and seeing how the user reacts to this action.

If the software has access to a GPS, the software should know in which country and city or region of the world the software is (longitude and latitude). The software should take a couple of GPS time measurements to determine whether it is moving and the speed of its movement.

If the software is able to determine all the information described above, then it can continue with assessing the kind of user (see step 3.1.2).

If one of the GPS devices is not working, it is a matter of concern depending on where the software is, but if all the GPS devices are unavailable, then this is a sign of a possible threat. It is possible that the software is in friendly territory, but in a location without GPS signal. The software should then analyze the previous GPS locations and their time of occurrence. If the previous analysis of the user was determined to be a friendly user or at a friendly location, then the software does not have to take any immediate action, but should stay alert. If on the other hand the previous detected user was non-friendly or at a non-friendly location, this should be considered as a possible threat and it should be acted accordingly.

If GPS is initially not available with the secure self-protected software, then the software must use other means to determine the location and type of user.

(1) The first option is for the self-protected software to go and look into the user file system (if such file system is available to our software).

   a. The names of the files could be a good lead to the user’s nationality (i.e., files with mostly Chinese names indicate with high probability a Chinese user).
   b. The contents of the files may also lead to the user’s nationality.

(2) If the software is running on a phone or laptop, ask the user to identify himself/herself.

   a. Ask for the user password for the use of the device.
   b. If the device has any biometric capture means, ask to provide such biometric data.

(3) Try the user’s internet connection if available, to gain access to user emails.

   a. The contents of the e-mails may help identify the user’s nationality.
   b. The headers of the e-mail may give the location of the user.

(4) If a direct internet connection is available, a message should be sent to a specific recipient to include all the captured information about the current location and the type of user threat. Software will delete all traces of the sent e-mail. Even when an answer is not expected back, the software has made the home base of the software aware of the situation and has given as much information as possible to be used for possible external actions or to improve the next generation of self-protected software.

(5) If a direct internet connection is not available, software should try to see whether it could establish communications through Wi-Fi.

(6) Software should try to communicate with any devices in close proximity *via* Bluetooth. If software can get into user's cell phone, software can obtain much information from this source.

(7) Software should analyze all the devices connected to software's computer [16].

a. The type of computer used may give software information about the user.
b. The type of keyboard used may lead to the user's nationality.
c. The type of printer. For example, a printer printing in Arabic is a good indication of the user's nationality or origin.

Note: All communications, searches and testing should be done in stealth mode; that is, no lights, noises or movements should be produced by these actions or at least they should be minimized.

Another form of attack is a GPS spoofing attack. These attacks attempt to deceive a GPS receiver by broadcasting incorrect GPS signals. The signals are structured to resemble a set of normal GPS signals or rebroadcast genuine signals captured elsewhere or at a different time. Even though the military GPS P-code is heavily encrypted and the Y-code encryption algorithm is not available to civilian users and would be difficult to spoof, the software should always monitor the GPS signals to detect and prevent spoofing. Wen et al. [17] described 10 different procedures to prevent GPS spoofing.

3.1.2 Attacker Threat Level

We use the four types of intruders defined in Gonzalez [18] and listed here:

"Casual-attacker: The attacker is not technically knowledgeable enough to do any reverse engineering and retrieve data or algorithms from the machine code software.

Hacker-attacker: This attacker is enough knowledge to access and retrieve some data and/or algorithms from the machine code sources of the software. Code development of the self-protected software should as a minimum include security procedures for attacks of this kind.

Institution-attacks: These are attacks sponsored by industry. They either hire somebody or use own resources to do the attack. This kind of attack is called industrial espionage.

Government-attack: The government of a nation through one or more of its agencies generates this attack and will use all the technical and legal resources available to such agencies."

It is important to notice that the main difference between the Gonzalez [18] previous work and this work is that in the previous work, there is Internet access by the software and a secure server exists to communicate with the standalone software, which is not the case with this current work, in which an Internet access is not needed and there is no secure server communication with the standalone software.

To determine the kind of threat the current user represents, the software has to evaluate all the information available about the user and the current location. Following is a list of situations that help the software determine the level of user threat:

(1) When the software detects that it has been modified (i.e., GPS routine was bypassed), then the threat is at least a hacker-attack. It is highly recommended to have at least two different locations inside the software where the software can check this detection of software modification.

(2) The software's current location indicates that the level is an institution attack or government attack. It is sometimes very difficult to differentiate between institution-attack level and government-attack level. Therefore, if this software is expected to survive a level of government attack, then even if the software has a level of institution attack threat, it should treat it as a level government-attack. If on the other hand the maximum level the software is trying to protect against is the level of institution-attack (not a national security threat), then the self-protected software reaction to the threat could be economically based (see step 3.3).

(3) When the computer running the self-protected software is connected to any foreign device (see step 3.1.1), then the threat should be an institution attack or a government attack.

(4) When the system has a heartbeat device and this device reports an anomaly, the threat should be an institution attack or a government attack.

(5) When the self-protected software is at a non-friendly location, the minimum assumed threat level is a hacker attack. The self-protected software may be designed to differentiate between non-friendly, bad-non-friendly and very-bad-non-friendly attacks and set the threat accordingly.

(6) When the self-protected software is at a friendly location, with no signs of tampering, then the user can be considered friendly and defined as a no-threat at this time.

3.1.3 How Long at Present Location?

The amount of time in the current location is very important.

(1) When the software has been at the current location for some time (e.g. a month) and it is a friendly territory and the user is a non-threatening user, then the software can be assumed to be currently safe.

(2) If the location is new or is a recent location, the software has to continue to analyze the user.

(3) If it is an old location, but the location and/or user were determined to be a threat or a possible threat, the software should then decide whether the time has come to take extra measures with this user (see step 3.2).

3.2 Analysis Phase

Analysis phase consists of analyzing whether the software should be at the present location, predicting future locations or assessing the location level of threat. Figure 3 shows the flow diagram for this phase.

3.2.1 Should the Software Be Here?

If the software has the option of handling missions, then the question to ask is whether the software is inside the mission parameters (i.e., the geographical area designated for the mission) or not. If the software does not have the mission option, then once the software has determined its location (as accurately as possible), the question is: Should the software be here?

(1) If the answer is no, then this is a bad location and the software must immediately continue to determine the attacker threat level (see step 3.2.2). The software may be at the right location (example, city and country), but the user may still be a foe.

(2) If the answer is yes, the software may be in the right location (example, city and country), but the user may still be a foe. With a foe user, the software must immediately asses the level of threat and act accordingly (see step 3.2.2). A friendly user in a good location takes the software to the beginning of the self-protected loop.

(3) There may be another answer for the posted question: the software is not sure. This may be possible for several reasons. The software may be on a ship and now it is on international waters. The software does not know if the ship is a friend or a foe. The software must investigate more before taking any action.

If the software is moving, it should obtain the rate of change (e.g., walking, car, ship, airplane, …etc.) and depending on the type of rate, it should determine the direction the software is heading. With the heading and the rate of change, the software will generate a set of predicted future locations. Then, the software should analyze some of the predicted future locations and take action depending on where these locations are and how soon the software is going to get there.

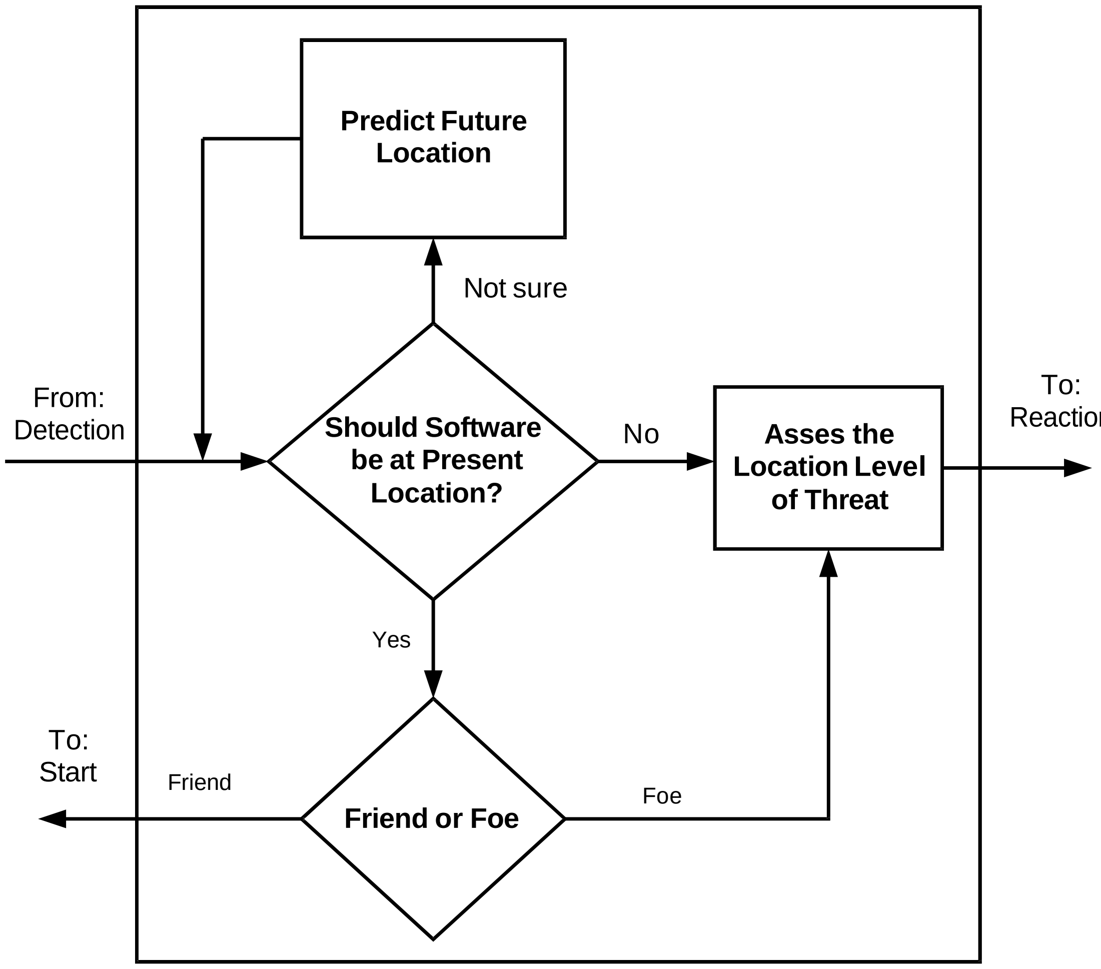

Figure 3. Analysis phase.

3.2.2 Assessing the Location Level of Threat

The question now is: How big of a threat is this current location or expected location in the near future?

At this point, the software may receive the following three kinds of scenario:

A friendly user in an unfriendly location. For this case, the self-protected software must stay alert, do nothing, but keep vigilant. An example of this kind of scenario is if the software is made in the U.S., then a friendly user may be an Australian user in a Venezuelan location.

A foe in a friendly location and a foe in an unfriendly location. For both cases of foe users, the software must go to the next phase and decide what to do next (step 3.3).

3.3 Reaction Phase

Reaction phase includes determining what actions to take and when to take such actions if any and finally perform actions. Figure 4 shows the flowchart of the process in this phase.

3.3.1 What Actions Should the Software Take?

The actions to take are going to change depending on the location, friend or foe and level of threat.

For a friend on a friendly location, only a signal is sent reporting any threat detected. History of threat and actions taken are saved.

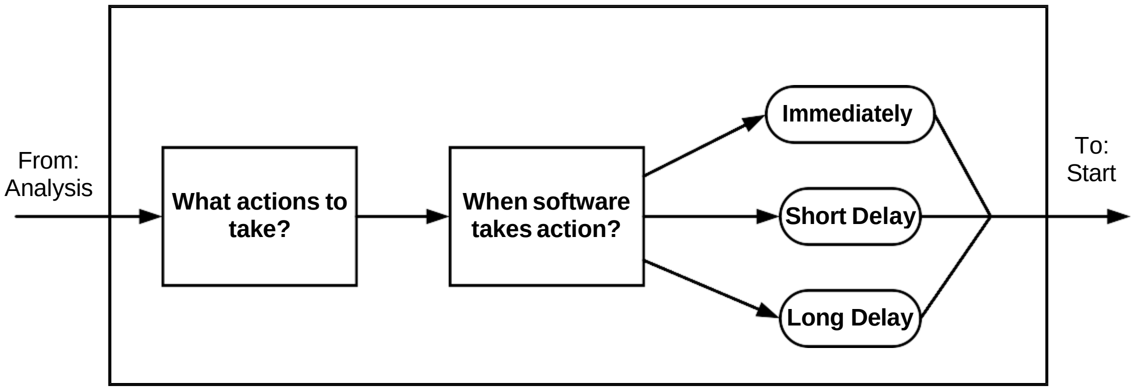

Figure 4. Reaction phase.

A friend on an un-friendly location should make the software heighten its awareness and make the software take more measurements (could be to add different ones or just increase the rate of taking measurements). This situation is not stable and can change very fast.

With any foe, the software should assess the level of threat and act accordingly.

For a casual attacker at any location, the author recommends taking all or some of the following actions:

(1) Delete and erase all files related to the self-protected software. If this is not possible, the system should try to re-encrypt all the self-protected software.

(2) If Internet is available, send a signal home (home, being a place in the Internet designed and operated by the government agency owner of the self-protected system) reporting the findings and actions taken.

(3) Erase or corrupt most of the user files.

(4) Display a message to the user stating that a malicious virus has taken control (the idea is to mislead him/her on the source of the problem and actions taken).

For a hacker attack when the software is at a friendly location and is guarding a maximum hacker attack, the author recommends taking all or some of the following actions:

(1) Perform same actions 1-3 of the casual attacker.

(2) Display a strong message to the user saying that his/her actions have been reported to the FBI, CIA, Interpol, …etc.

If we have the case that the software has a hacker attack and it is guarding against an institution attack or a government attack or the software is at an un-friendly location, the author proposes that the system performs the following actions and such actions be executed covertly.

(1) Modify and change the self-protected software (or its data) slightly (these actions should be decided and determined at the design time of the secure self-protected software); so it produces results, but the wrong results.

(2) Insert a malicious virus that spreads to all the contacts of this user and all nodes of the network used by this computer.

(3) Start the processes to destroy the local equipment. The due time could be up to a couple of days in order to give time in case the status of the threat changes, like moving to a friendly location. If a physical bomb is not possible (because of physical limitations or the law does not permit such action), then at the due time destroy as much software and information as possible. The destruction of the local equipment should never include human lives. Before any destruction is done to software, equipment or general area, it is necessary to be sure that this is not a false-positive. It is recommended to have fail-safe mechanisms to make sure that when such drastic action is required, we have a high probability of certainty.

For an institution attack (all covert actions):

(1) At friendly locations: Same actions 1-3 of casual attacker.

(2) For un-friendly locations: Same as hacker attack at an un-friendly location, except that the physical bomb due time is extended for a longer period (six months to a year). The idea is to do as much damage as possible.

For a government attack (all covert actions):

(1) Same actions as 1-3 of casual attacker.

(2) Insert a malicious virus that spreads to all the contacts of this user.

(3) Detonate a software bomb to destroy as much software and information as possible.

(4) Detonate a physical bomb for the destruction of the local equipment only. If this is a guard for government attack, this option should always exist when the software is delivered with the hardware included.

(5) Detonate a physical bomb for a larger area at the current location. This action may end up costing human lives. There is always the option of giving a warning to the current user before the action is taken. The decision to have this option installed in the system will be up to de owner of the software. This may not always be possible if no hardware is delivered with the software.

In all cases, the system should be aware of the legal principle of ‘fair labelling’ [19], which requires that the label of the offence (e.g. braking and modifying the self-protected software) should fairly express and signal the wrongdoing of the accused, so that the stigma (and receiving actions) of conviction corresponds to the wrongfulness of the act.

3.3.2 When Should the Software Take Action?

If the level of threat is determined to be a government attack, then, independently of the location, the actions to take must be immediate.

If the level of threat is an institution attack and the software is at a non-friendly location, then the actions will be taken immediately. On the other hand, if the software is at a friendly location, the author suggests a delay of the software bomb deployment.

For casual attacker and hacker-attack threat levels at friendly locations, the actions are immediate. For hacker-attack threats in non-friendly locations, the author suggests the delay of the software attack to the local computer for a couple of days, in order to make sure that the location is correct or to make sure that the threat level doesn’t increase.

If the level of threat is an institution attack and the software is at a friendly location, the actions are taken immediately. For non-friendly locations, it is suggested for example to delay in the software bomb deployment.

4. LEGAL ASPECTS

Over the last twenty years, many international lawyers have successfully crafted an elaborate and operational system of international criminal law [19]. The project used precepts of criminal law, international human rights and humanitarian law. The last two areas provided essential normative content and a familiar framework for international oversight and intervention.

Like most international issues, each country has its own way of interpreting the international law, especially when dealing with cyber war, covert actions or surveillance. In a major story [20], Yahoo News disclosed the existence of a 2018 presidential covert action finding altering the terms on which the CIA can (and should) engage adversaries *via* cyber means. The story is an important reminder that the CIA continues to play a critical role in the increasingly fierce gray zone competition that characterizes statecraft in cyberspace these days.

One of the most interesting points in the article involves a rule change that apparently removed or weakened a prior prohibition on operations that might “damage critical infrastructure.” Thus, a blank check to blow up critical infrastructure would indeed be deeply concerning. But it is possible—indeed, probable—that the finding does not give such a blank check. Possibly it just authorizes prepositioning of capabilities, in the event an adversary takes such an action against U.S. critical infrastructure (in our case it will be against the self-protected software) and thus opens the door to such a countermeasure.

At the current time [21], the development of information warfare presents international legal issues that will complicate nation’s efforts both to execute and to respond to certain information warfare attacks, specifically those using computers, telecommunications or networks to attack. We have described actions to take when the self-protected software is under attack, but the legality of such actions is left to the owner and designer of the self-protected software.

Information warfare weapons must meet the same tests for necessity and proportionality as other weapons under the laws of armed conflict [22]. In addition, commanders must recognize and weigh the possible consequences of weapons that can devastate the information systems of an adversary. Problems such as lack of enemy command and control, post-hostility reconstruction and retaliation, among others, must be considered by the commander contemplating the use of information weapons. Because of the extraordinary consequences of these weapons, developers must provide guidance for their employment and commanders must carefully consider adverse effects from their use.

In the field of cyber security, ill-defined concepts and inconsistently applied terminology are further complicating an already complex issue [23]. This causes difficulties for policy-makers, strategists and academics. Using national cyber security strategies to support current literature, this paper undertakes three tasks with the goal of classifying and defining terms to begin the development of a lexicon of cyber security terminology. The term “active defence” is common in the military as the idea of offensive action and counterattacks to deny advantage or position to the enemy. The concept remains elusive when applied to the cyber domain and suffers a lack of clarity in related law and national policy.

We want to include in this section issues related to the international law interpreted by Russia and China.

To understand how the Russians interpret international law, we recommend the book by Mälksoo [24]. This book addresses a simple question: how do Russians understand international law? Is it the same understanding as in the West or is it in some ways different and if so, why? It answers these questions by drawing on from three different yet closely interconnected perspectives: history, theory and recent state practice.

For the Chinese interpretation of international law [25], this article highlights a different set of elements that become manifest in assessing the rapid overall rise in references to and application of international law by courts in China in recent years.

5. Analytical Model

Let S be the software that is protected without user detection

Let SD be the software that is protected including user detection

Thus SD=S + {user detection}

Let L be an intruder type with possible values of L1 for Casual-attacker or L2 for Hacker-attack or L3 for Institution-attack or L4 for Government-attack intrusion types

Then L=L1 ∧ L2 ∧ L3 ∧ L4

EF(S,I) the effort (in man-hours) for intruder I to break the software protection and reverse engineer the software S where I \in L

In general, we define EF(S,L1) > EF(S,L2) > EF(S,L3) > EF(S,L4)

Theorem: $\boldsymbol{EF(S,I) \leq EF(SD,I)\ \forall I \in L}$

Proof:

Let us assume that a user (intruder) is trying to reverse engineer the software

The effort (in man-hours) used trying to break software S by intruder I at time t is EB(S,I,t)

$$\text{BREAK(S,I,t)} = \begin{cases} \mathit{true} & \text{if software S was broken by intruder I at time t} \\ \mathit{false} & \text{otherwise} \end{cases}$$

Let's call t_b the time at which BREAK(S,I,t)= true

and t=0 the time at which the intruder started trying to break the software.

Thus **EF(S,I)=** $\sum_{t=0}^{t=t_b} \mathbf{EB(S, I, t)}$

If we have user detection, then we can assume then at some time t, t>0 our user detection algorithm detects the type of user and determines the intrusion type

Let us call this time of user detection t_d

We have three options:

$t_b > t_d$,

$t_b = t_d$

$t_b < t_d$,

For $t_b > t_d$, at time t_d BREAK(SD,I, t_d)= false, then at this time our user detection algorithm will take an action accordingly to the type of intrusion threat. This action will by definition make the effort of the intruder to obtain a software break much greater.

Thus EF(S,I) < EF(SD,I)

For tb = td, the software is broken at the same time of the user detection. This will be a low probability type of occurrence because of the exactness of the equality.

Thus EF(S,I) = EF(SD,I).

For $t_b < t_d$, the software was already broken by the time of the user detection. This is the worst case, since the intruder was able to reverse engineer our software. With user detection, we will be able to retaliate for this break and cause some damage to intruder I. We expect that any repairs to the damage caused by our user detection algorithm will generate an effort for the intruder.

Thus EF(S,I) = EF(SD,I)

If we add the effort to repair the damage, then we will add a positive number to our EF(SD,I); that is:

The real EF(SD,I) = old EF(SD,I) + {effort to repair retaliation}

And since the {effort to repair retaliation} is always a positive number, then

EF(S,I) < EF(SD,I).

We have shown that under all circumstances, the final effort to break the software is equal or greater if we use user detection inside our software.

This proves our theorem Q.E.D.

Thus, having user detection inside our software will always be better than not having it.

6. Recommendations and Conclusions

6.1 Main Conclusion

We have proved and concluded that having and using user detection in self-protected software is a means to increase the security of such software. Thus, the main contribution of this article is the description of a process methodology to follow when designing military secure self-protected software. The set of guidelines described in this paper to detect location, size of threat and user type are not meant to be exhaustive.

6.2 Recommendations and Discussion

The topic of this paper is highly sensitive for military; therefore, almost all research papers on this topic are classified as secret or top secret. Currently, there is nothing in the un-classified literature about this topic.

The set of actions to take in response to the different threats to some people may seem to be over the top, but the author firmly believes that in some cases, we have to react as if we were at war. If what we are protecting is of high security value and it will be very negative for our country to have our enemies having or using this Self-Protected Software System, then our actions reflect this belief. Our enemies will not hesitate to illegally use our resources even if this causes economic loss or loss of human lives. This set of actions is defined at the start of our software development life cycle by the owners of the software, which at the end are responsible (morally, economically and legally) for the actions taken by the secure self-protected software. On the other hand, we do not want our software to make mistakes, giving us false-positives and doing unnecessary harm.

We recommend the use of safety standards and methodologies for the development of this self-protected software [26]-[29].

It is important to point out that because of the inherent mobility of cell phones and laptops, the handling of self-protected software security should involve continuous authentication of the user's mobile device [30] along with user detection. It is more probable that a change of user occurs on a mobile device than any other device, because mobile devices are easy to steal, borrow, forget, misplace, …etc.).

When analyzing the user's threat level which is related to the software risk-level:

Casual attacker: The software is only protected for casual attackers. A minimum of security is needed.

Hacker attack: The software is protected against hacker attacks. This intruder may or may not have initial economic gain plans for the intrusion. In most cases, it is the intellectual challenge that motivates this intruder (i.e., hacker), but economic gains may not be very far behind.

Institution attack: The software is protected against institutional attacks. Economic gains are the main reason for the intrusion. In most cases, with enough time and money, any secure self-protected software may be cracked. Therefore, the developing team should always work with the goal of making the intruder's effort needed to break the secure code high enough and not cost-effective.

Government attack: The software is protected against government attacks. In most cases, with enough time and money, any secure self-protected software may be cracked. It is recommended that techniques for intruder detection [31], [3], [6], [32]-[34] (if an Internet connection is available) and the user detection described in this paper with the respective actions to take (covert and not-covert) be included in the secure code. This level of protection requires the use of the most sophisticated security algorithms.

Our software has to be aware of the level of threat it is supposedly guarding against. For hacker attack and institution attack, the main objective of the threat is economic. Therefore, our reaction to a threat of this kind could end up causing the user economic loss. These actions could vary from erasing all software from the machine to releasing a malicious virus to providing wrong output in the software output.

In general, to combat secure self-protected software threats, organizations must adopt pre-emptive and reactive measures by building self-defending mechanisms into their software, so that their application is self-protected on the device. Finally, to make the security of the software complete and more robust, we should obfuscate [35] our secure self-protected software.

The protection and security of self-protected software constitute an ever-present concern for the military agencies of any government producing their own hardware-software military equipment that is mobile. Therefore, the techniques described in this paper should be required reading for industries producing for the military any self-protected software.

For our future work, we are planning to do a simulation of the suggested methodology and we will use AlSobeh [36] framework to do the simulation.

REFERENCES

[1] A. Deeks, "An International Legal Framework for Surveillance," Virginia Journal of International Law, HeinOnline, 2014.

[2] S. H. Amer and J. A. Hamilton, "Intrusion Detection Systems (IDS) Taxonomy: A Short Review," Journal of Software Technology, vol. 13, 2010.

[3] T. F. Lunt, "A Survey of Intrusion Detection Techniques," Computers & Security, vol. 12, no. 4, pp. 405–418, 1993.

[4] Y. Zhang, W. Lee and Y. Huang, "Intrusion Detection Techniques for Mobile Wireless Networks," Mobile Networks and Applications (Georgia Institute of Technology), pp. 1–16, [Online], Available: http://wenke.gtisc.gatech.edu/papers/winet03.pdf, 2003.

[5] M. Koch and K. Pauls, "Engineering Self-protection for Autonomous Systems," Proc. of the International Conference on Fundamental Approaches to Software Engineering (FASE 2006), Part of the Lecture Notes in Comp. Sci. Book Series, vol. 3922, pp. 33-47, DOI: 10.1007/11693017_5, 2006.

[6] Y. Al-Nashif, A. A. Kumar, S. Hariri, G. Qu, Y. Luo and F. Szidarovsky, "Multi-level Intrusion Detection System (ML-IDS)," Proceedings of the IEEE International Conference on Autonomic Computing (ICAC'08), pp. 131-140, Chicago, USA, 2008.

[7] A. Elkhodary and J. Whittle, "A Survey of Approaches to Adaptive Application Security," Proceedings of the Workshop on Software Engineering for Adaptive and Self-managing Systems (SEAMS'07), DOI: 10.1109/SEAMS.2007.2, Minneapolis, USA, 2007.

[8] A. Thakkar and R. Lohiya, "A Review of the Advancement in Intrusion Detection Datasets," Procedia Computer Science, vol. 167, pp. 636–645, 2020.

[9] K. Guercio, "Best Intrusion Detection and Prevention Systems for 2021: Guide to IDPS," eSecurityPlanet, [Online], Available: https://www.esecurityplanet.com/products/intrusion-detection-and-prevention-systems/, 2021.

[10] M. S. Ben Mahmoud, N. Larrieu, A. Pirovano and A. Varet, "An Adaptive Security Architecture for Future Aircraft Communications," Proceedings of the 29th Digital Avionics Systems IEEE Conference (DASC), DOI: 10.1109/DASC.2010.5655363, Salt Lake City, USA, 2010.

[11] K. John, "BAE Systems to Install Electronic Warfare (EW) Self-protection Pod to Help Defend P-8A Poseidon Aircraft," Military & Aerospace Electronics, [Online], Available: https://www.militaryaerospace.com/communications/article/14195763/electronic-warfare-ew-aircraft-selfprotection, 2021.

[12] J. LaPadula Leonard, "Intrusion Detection for Air Force Networks," Mitre Technical Report, MTR 97B0000035, October 1997.

[13] A. Jay, "Intrusion Detection Systems: The First Line of Defense," SCIF Global Technologies, [Online], Available: https://scifglobal.com/intrusion-detection-systems-the-first-line-of-defense/, 2015.

[14] J. P. Mello Jr., "What is Runtime Application Self-protection (RASP)?," TechBeacon, [Online], Available: https://techbeacon.com/security/what-runtime-application-self-protection-rasp application-self-protection-a-must-have-emerging, 2016.

[15] J. Lavery, "The Future Is Now: Applications Protect Themselves against Attacks," Veracode, [Online], Available: https://www.veracode.com/blog/2016/06/future-now-applications-protect-themselves-against -attacks, 2016.

[16] S. Giehl, "Device-detector," Github, [Online], Available: https://github.com/matomo-org/device-detector, 2021.

[17] H. Wen, P. Y.-R. Huang, J. Dyer, A. Archinal and J. Fagan, "Countermeasures for GPS Signal Spoofing," Proceedings of the 18th International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GNSS 2005), pp. 1285-1290, Long Beach, CA, 2005.

[18] C. Gonzalez, "Adaptive Standalone Secure Software," US Patent 10,521,613, B1, [Online], Available: https://patentimages.storage.googleapis.com/eb/fb/4b/82980dc0f04d32/US10521613.pdf, Dec. 2019.

[19] D. Robinson, "The Identity Crisis of International Criminal Law," Leiden Journal of International Law, vol. 21, pp. 925–963, 2008.

[20] R. Chesney, "The CIA, Covert Action and Operations in Cyberspace," Lawfare, [Online], Available: https://www.lawfareblog.com/cia-covert-action-and-operations-cyberspace, July 2020.

[21] L. T. Greenberg, S. E. Goodman and K. J. Soo, Information Warfare and International Law, National Defense University Press, 1998.

[22] M. K. Kuschner, "Legal and Practical Constraints on Information Warfare," [Online], Available: https://www.airuniversity.af.edu/Portals/10/ASPJ/journals/Chronicles/kuschner.pdf.

[23] R. S. Dewar, "The Triptych of Cyber Security: A Classification of Active Cyber Defence," Proc. of the 6th International Conference on Cyber Conflict (CyCon), DOI: 10.1109/CYCON.2014.6916392, 2014.

[24] L. Mälksoo, Russian Approaches to International Law, Oxford, ISBN-13: 978-0198808046, 2015.

[25] C. Cai, "International Law in Chinese Courts during the Rise of China," American Journal of International Law, vol. 110, no. 2, pp. 269-288, DOI:10.5305/amerjintelaw.110.2.0269, 2016.

[26] IEEE, "IEEE Standard for Software Safety Plans," IEEE Standards Association, IEEE 1228-1994, August 1994.

[27] Joint Software System Safety Committee, "Software System Safety Handbook: A Technical & Managerial Team Approach," US Department of Defense, [Online], Available: https://dl.icdst.org/pdfs/files/42fd057643931936afc1e649cee8c723.pdf, Dec. 1999.

[28] N. G. Leveson, Safeware: System Safety and Computers, Addison-Wesley, 1995.

[29] MIL-STD-882E, "Department of Defense Standard Practice: System Safety," US Department of Defense, May 2012.

[30] C. Gonzalez, "Methods and Apparatus to Provide and Manage Security for the Access to Mobile Electronic Devices," US Patent, Patent no. US 7,941,669 B2, [Online], Available: https://patentimages.storage.googleapis.com/46/58/d5/bbc2a56707980d/US7941669.pdf, March 2015.

[31] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Macia-Fernandez and E. Vazquez, "Anomaly-based Network Intrusion Detection: Techniques, Systems and Challenges," Comp. & Sec., vol. 28, pp. 18–28, 2009.

[32] D. Frincke, A. Wespi and D. Zamboni, "From Intrusion Detection to Self-protection," Comput. Netw., vol. 51, no. 5, pp. 1233-1238, [Online]. Available: https://doi.org/10.1016/j.comnet.2006.10.004, 2007.

[33] A. Nagarajan, Q. Nguyen, R. Banks and A. Sood, "Combining Intrusion Detection and Recovery for Enhancing System Dependability," Proceedings of the IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W), pp. 25-30, Hong Kong, China, 2011.

[34] N. Stakhanova, S. Basu and J. Wong, "A Taxonomy of Intrusion Response Systems," Int. J. Inf. Comput. Sec., vol. 1, no. 1, pp. 169—184, 2007.

[35] J. J. Hagg, "A Simple Introduction to Obfuscated Code," Dream.In.Code, [Online], Available: http://www.dreamincode.net/forums/topic/38102-obfuscated-code-a-simple-introduction/, Sep. 2015.

[36] A. AlSobeh, S. AlShattnawi and A. Jarrah, "WEAVESIM: A Scalable and Reusable Cloud Simulation Framework Leveraging Aspect-oriented Programming," Jordanian Journal of Computers and Information Technology (JJCIT), vol. 06, no. 02, pp. 182-201, June 2020.

ملخص البحث:

الهدف من هذا البحث وصْف منهجية عملياتية لزيادة الأمان داخل البرمجيات العسكرية الآمنةذاتية الحماية. إنّ البرمجيات ذاتية الحماية هشّة للتَّهديدات، ويعتمد ذلك في الغالب على مُستخدم تلك البرمجيات. لذا، فإن كشف المستخدم الحالي للبرمجيات ذاتية الحماية أمر شديد الأهمية.

تشمل المنهجية المقترحة على ثلاث مراحل: كشْف المستخدم، وتحليل الحالة الرّاهنة، وردود الفعْل. وتتكون مرحلة الكشف من تقدير الموقع الجغرافي، والوقت في الموقع الرّاهن، وتحديد نوع المستخدم (صديق أو عدوّ). أما مرحلة التحليل، فتشمل تحليل ما إذا كانت البرمجيات ذاتية الحماية ينبغي أن تكون في الموع الحالي، وتوقّع المواقع المستقبلية، وتقدير مستوى التهديد. وتتضمن مرحلة ردود الفعل تحديد الأفعال الفورية أو المتأخرة –إن وجدت– وتنفيذها بناءً على الحالة.

كذلك تم التّطرق الى المسائل القانونية المتعلقة بهذا الموضوع، ووصْف الإجراءات المضادّة وردود الفعل المقترحة. ويبين النّموذج التحليلي أنّ البرمجيات ذاتية الحماية التي تحتوي على خاصية كشْف المستخدم توفّر حمايةً أكبر مقارنة بمثيلاتها التي لا تحتوي على هذه الخاصية.