SpyCon: Emulating User Activities to Detect Evasive Spyware

M.Chandrasekaran, S.Vidyaraman and S.Upadhyaya
Computer Science and Engineering
University at Buffalo
Buffalo, NY 14260
{mc79, vs28, shambhu}@cse.buffalo.edu

Abstract

The success of any spyware is determined by its ability to evade detection. Although traditional detection methodologies employing signature and anomaly based systems have had reasonable success, new class of spyware programs emerge which blend in with user activities to avoid detection. One of the latest anti-spyware technologies consists of a local agent that generates honeytokens of known parameters (e.g., network access requests) and tricks spyware into assuming it to be legitimate activity. In this paper, as a first step, we address the deficiencies of static honeytoken generation and present an attack that circumvents such detection techniques. We synthesize the attack by means of data mining algorithms like associative rule mining. Next, we present a randomized honeytoken generation mechanism to address this new class of spyware. Experimental results show that (i) static honeytokens are detected with near 100% accuracy, thereby defeating the state-of-the-art anti-spyware technique, (ii) randomized honeytoken generation mechanism is an effective anti-spyware solution.

Keywords

Associative Rule Mining, Honeytokens, Spyware, User Activity

1. Introduction

Potentially Unwanted Programs (PUPs) is the collective term given to programs whose presence poses a serious security and privacy threat to users [17]. This includes malware, spyware, adware and other myriad programs. The commercial incentives of these programs are lucrative enough for this 'industry' to thrive, and according to some projections [17], are expected to rise at exponential rates in the future.

The success of any spyware on a system is determined by its ability to evade detection. Towards this goal, early spyware had the advantage of user ignorance and lack of security mechanisms or tools to detect and remove them. Since then, various anti-spyware mechanisms like toolbars, various detection and removal tools, etc., have been developed. These defense solutions employ either signature based or anomaly detection (flow based [5, 6]) philosophies. Even though signature based systems have the advantage of detecting known spyware programs with a high degree of accuracy, they are incapable of detecting novel threats. Anomaly detection schemes, on the other hand, are capable of detecting new threats with reasonable accuracy. They operate on the premise that any behavior observed in a system that deviates from the 'normal' behavior is indicative of the presence of unauthorized actions.

But, as the defenses mature, so do the attacks. Spyware authors have responded to these defense mechanisms by altering their code to create new families and variants that are either incremental updates or binary obfuscation of earlier code. Such 'new' spyware has an inherent advantage against signature based analysis since a known profile of spyware is not immediately available for comparison. Hence the major threat for new spyware is primarily from anomaly detection systems, which compare the normal behavior of systems (based on parameters like web sites visited, network connections initiated, etc.) to the behavior of the system infected with spyware. To counteract these anomaly detection schemes, current spyware programs attempt to blend in with legitimate behavior. For example, a spyware may contact its remote home servers only when it detects user activity, thereby blending in with the 'normal' profile.

To address these kinds of threats, Borders et al. [6] propose a system where a local agent would generate a sequence of honeytokens. Honeytokens [23] are defined as "*honeypots that are not computer systems"* whose value *"lies not in their use, but abuse.*" In the context of this paper, a set of network requests generated by the local agent (to emulate legitimate user activity) represent honeytokens. The idea behind this agent is that spyware, if present, would operate when the local agent generated the honeytoken, thereby giving itself away. For example, a spyware may operate only when it detects user activity and remain passive otherwise. In [6], the local agent called *Siren* generates honeytokens that mimic user activity during the idle pe-

riods; spyware operates assuming the user is active and hence can be caught by a network based IDS. This is the current state-of-the-art in terms of spyware detection.

Given the history of spyware creation and operation, it is quite likely that the next update of spyware will attempt to bypass this mechanism too. The work in this paper presents a methodology whereby simple mechanisms like honeytoken generation as in [6] can be detected by a new class of spyware called *SpyZen*. The basic concept behind our scheme is quite intuitive. *SpyZen* spyware operates in three stages. The first stage is an '*install and observe'* stage, where spyware merely listens to the sequence of events, of which certain portions are honeytokens. In the second stage of *analysis and inference*, spyware detects the honeytoken sequence. Using data analysis algorithms like Associative Rule Mining algorithm [4], spyware can infer the honeytoken generation. The third stage is the actual *operation* stage, where spyware operates only when the honeytoken sequence is not detected. We then present a defense mechanism against this new class of spyware, called *SpyCon*, which utilizes a randomized honeytoken generation scheme.

Thus, this paper takes as its point of departure the work by Borders et al. [6], where the authors present a honeytoken based defense mechanism against spyware that operates only when there is user operation (thereby masking its network access under the guise of user operation).

1.1 Summary of Contributions

The contributions of the paper can be summarized as follows:

- We first propose a new class of 'intelligent' spyware called *SpyZen* that not only bypasses anomaly detection systems like [5], but also actively defeats honeytoken based schemes [6].
- Secondly, we present a defense technique called *SpyCon* that builds on prior works [6] to detect the class of *SpyZen*-like threats. By devising *SpyZen*, our work brings out the weakness of *Siren* and *SpyCon* demonstrates how the honeytoken based schemes can be strengthened against novel attacks.

The threat model of the *SpyZen* class of spyware is described and their capabilities are clearly delineated.

The rest of the paper is organized as follows. Section 2 discusses the related work and places our work in perspective. Section 3 defines the problem addressed in this paper. Section 4 presents the design of *SpyZen*, the new class of spyware that can defeat anti-spyware techniques that rely on static honeytoken generation. The new and improved randomized honeytoken generation algorithm is presented in Section 5. The experimental details are presented in Section 6. Section 7 discusses some practical implementation issues. Concluding remarks and future work are presented in Section 8.

2. Related Work

Anti-spyware efforts has attracted a lot of attention in recent years, partly due to the popularity of the web and observed increase in spyware programs. Sariou et al. [22] provide a comprehensive summary of spyware activity in a university environment. Moshchuck et al. [19] extend this one step further and conduct a crawler based study to analyze the prevalence of spyware programs on the web. In the study, they reveal that 13.4% of 21,200 executables downloaded from 18 million web pages were infected with spyware programs. As a first line of defense, traditional approaches focused on extending anti-virus and anti-malware solutions to shield against spyware-related threats. However, spyware programs growing in sophistication and ingenuity render these solutions weak. The current anti-spyware efforts can be broadly classified into the following categories based on their underlying detection technique: (i) programming language based detection; (ii) behavior based detection; and (iii) signature based detection.

Programming language based approaches using static analysis and run-time code checking have been proposed to detect malicious executables. Christodorescu et al. [8] employ static analysis to detect malicious patterns in executables that belong to a particular malware class. However, as such schemes rely on finding malicious patterns in executables, they can be bypassed using simple program obfuscation techniques such as [2]. To detect polymorphic and metamorphic variants, [9] proposes to include the semantics of the executables' instruction set as additional information to guide static analysis. Here, the set of instructions depicting malicious behavior is provided as templates, and the detection engine operates to check if the corresponding program actually exhibits the specified malicious behavior. While being effective in detecting simple malware variants, these techniques incur increased performance cost, and are not feasible in all circumstances [20, 15].

Behavior based techniques capture the working of applications in their operating environment. This information is used to monitor for deviation from the specified normal behavior. Microsoft's Gatekeeper [26] is a behavior based technique which uses *auto-*

start extensibility points (ASEP) to check if the programs loaded on system boot were installed automatically without user invocation. Similarly, Kirda et al. [11] propose a behavior based detection technique to detect spyware programs that hook themselves with the browser interface. Responses of browser add-ons (also called browser helper objects (BHO)) to simulated browser events are analyzed to check if the exhibited behavior is malicious or not. Despite the effectiveness of the abovementioned behavior based techniques, these techniques are not generic and confine to one particular operating environment, i.e., Microsoft Windows with Internet Explorer. An orthogonal approach uses flow-based detection to analyze outbound web requests for detecting spyware programs which furtively leak out user information to remote servers. Webtap [5] is one such tool which monitors outbound HTTP traffic to detect spyware programs by separating the user activity from the spyware activity. However, as recent spyware programs blend with the user activity to evade detection, distinguishing spyware activity from the user activity is not always possible. To detect such evasive spyware, a honeytoken based approach [6] has been proposed. These techniques operate by sending a known sequence of network requests that mimic user activities. As spyware cannot distinguish honeytokens and legitimate user activity, they attempt to operate with the honeytokens by making additional network requests (which are then identified by the gateway NIDS). Unfortunately, if static honeytokens are used, it becomes trivial to design new class of intelligent spyware that can evade detection. To this extent, in this paper, we design *SpyZen* as a proof-of-concept spyware that circumvents the static honeytoken based approach. In addition, as an extension we propose *SpyCon*, a randomized honeytoken generation technique to eliminate *SpyZen* threats.

Finally, commercial tools [1, 3, 18] exist which examine hard drives periodically to detect the presence of spyware programs. However, as these techniques are signature-based and scan for malicious patterns in files or registry locations, they fail to protect against novel spyware for which no signatures are available yet. As we adopt flow-based detection, our technique can be easily extended to protect from both seen and unseen spyware variants.

3. Problem Definition

Consider the normal activity of the user on a local host. This could be network access requests, etc. An external Network based Intrusion Detection System (NIDS) and a local agent operate in conjunction to monitor and classify the normal operation of the local user.

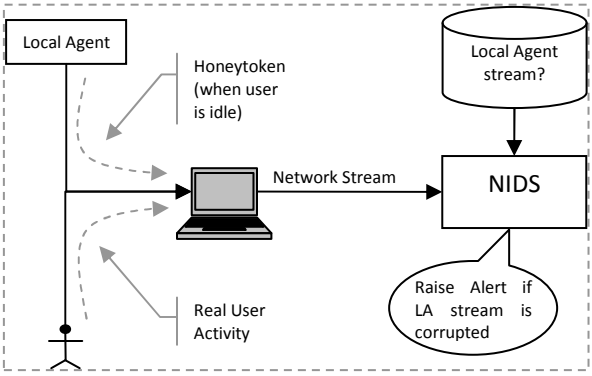

Figure 1. Problem Formulation Scenario

As shown in Figure 1, the Local Agent (hereafter referred to as the LA), generates a honeytoken (a sequence of network access requests) that is known to the NIDS. This generation is initiated only if there is no user activity. The first problem, from the viewpoint of a spyware, is to differentiate between real user activity sequence and the honeytoken sequence. The second (and the main) problem, from the viewpoint of the LA, is to generate a sufficiently random honeytoken sequence so as to prevent spyware from detecting the honeytoken. This was suggested as an open problem in [6], where the authors aptly named it a *passive reverse Turing test*. Although the methodology suggested here does not solve the open problem *per se*, we do show the inadequacy of deterministic honeytoken generation and suggest randomized algorithms for better generation.

3.1 Problem Formulation

Towards constructing a formal representation of the problem, we define the following terms.

Definition 1. A Network honeytoken $\mathbf{N^h}$ is defined as a sequence of n network access requests $\mathbf{N^h} = \{N^h_{req\text{-}1}, N^h_{req\text{-}2}, \ldots N^h_{req\text{-}n}\}$ where the following properties hold true:

- $\forall\ 1 \le i \le j \le n$, $time(N^h_{req\text{-}i}) < time(N^h_{req\text{-}j})$
- $\forall\ 1 \le i \le j \le n$, the sequence between $N^h_{req\text{-}i}$ and $N^h_{req\text{-}j}$ is known/predetermined

Definition 2. A user network activity $\mathbf{N^u}$ is defined as a sequence of n network access requests $\mathbf{N^u} = \{N^u_{req\text{-}1}, N^u_{req\text{-}2}, \ldots N^u_{req\text{-}n}\}$ where

- $\forall\ 1 \le i \le j \le n$, $time(N^u_{req\text{-}i}) < time(N^u_{req\text{-}j})$
- $\forall\ 1 \le i \le j \le n$, there exists no relation between $N^u_{req\text{-}i}$ and $N^u_{req\text{-}j}$

Definition 3. A recorded network access pattern $\mathbf{N^r}$ is a mixed sequence of network accesses $\mathbf{N^r} = \{N^r_{rec\text{-}1}, N^r_{rec\text{-}2}, \ldots, N^r_{rec\text{-}n}\}$ where:

- $\forall\ 1 \le i \le j \le n$, $\text{time}(N^r_{\text{rec-}i}) < \text{time}(N^r_{\text{req-}j})$
- $N^r_{\text{rec-}i} \in \{\mathbf{N^h}, \mathbf{N^u}\}$

Thus, the recorded sequence is a mixture of the honeytoken sequence and legitimate user activity. The problem, therefore, is to separate a given recorded network access pattern (in real time, as a stream) into its constituent components, viz. the honeytoken sequence and the legitimate user activity. Our Problem can be defined in terms of the following algorithm.

Problem 1: Devise an algorithm Algo-Find-Honeytoken(A-FH) such that:

$$\text{A-FH}(\mathbf{N^r}) = \{\mathbf{N^h}, \mathbf{N^u}\}$$

where the input is a recorded network access pattern $\mathbf{N^r}$ and the output is the mixed sequence decomposed into $\mathbf{N^h}$ and $\mathbf{N^u}$.

However, this requires that we identify not only the honeytoken sequence, but also the user activity. Furthermore, this precludes other network streams like automatic software updates (windows updates), toolbar activities (Google toolbar), etc. From the viewpoint of spyware, what is required from the recorded network stream is the extraction of only the honeytoken sequence, so that spyware may refrain from operating in those durations. This leads us to the practical problem formulation defined in Definition 2.

Problem 2: Devise an algorithm Algo-Find-HoneyTokenTime(A-FHT) such that:

$$\text{A-FHT}(\mathbf{N^r}) = \{\mathbf{N^h}\}$$

Hence, spyware can stop operating from t_1 through t_n where $t_1 = \text{time}(N^h_{\text{req-1}})$ and $t_n = \text{time}(N^h_{\text{req-n}})$. Note that A-FHT($\mathbf{N^r}$) will defeat the spyware detection.

4. Design of *SPYZEN*

Our goal is to show the inadequacy of static honeytoken based anti-spyware techniques by creating a new generation of spyware, *SpyZen*, by slightly modifying the existing spyware's operation. As given in the previous section, the effectiveness of honeytoken based anti-spyware mechanisms depend on their ability to mask the honeytoken sequence so that it is hard for the spyware programs to deduce them. However, since static honeytoken based approaches employ the same sequence of network requests repeatedly, the probability of correlating a sequence of network requests to $\mathbf{N_h}$ becomes very large. Hence, a frequent observation of the same set of network requests is a strong indication of occurrence of honeytoken sequence.

According to the definition of evasive spyware, the spyware programs leak out users' information only in the presence of user activity. Here, we limit the scope of the threat vector by assuming that the presence of user activity can be perceived only by observing the network requests made, and not by other schemes such as keystrokes and mouse movements monitoring. Furthermore, given the characteristics of static honeytoken, we design *SpyZen* to infer honeytoken sequence $\mathbf{N^h}$ from observed network requests $\mathbf{N^r}$ using associative rule mining, which is well-suited for mining frequent itemsets and relationships that exist between them in large samples of data.

As a simple and powerful tool, associative rule mining has been employed in many areas of security such as intrusion detection [10, 13, 14, 16] and spam email detection [25]. Associative rule mining is used for finding relationships between the occurrences of itemsets within transactions, i.e., given a set of items and a set of transactions, rules are generated which link the otherwise disjoint itemsets. Associative rule mining was first proposed by Agrawal et al. [4] for market basket data or transactional data analysis to determine which items are most frequently purchased together.

In the remainder of this section, we briefly describe the basic concept of associative rule mining, then present *SpyZen*, and show how associative rule mining can be applied to A-FHT.

4.1 Associative Rule Mining: An Overview

Given a set of disjoint items ($\mathbf{I}$) and a set of transactions ($\mathbf{T}$), where each transaction consists of a subset of the set of items, associative rule mining is used to determine relationships that exist between the occurrences of items within transactions. Translating into mathematical terms, given a set of items $\mathbf{I}$, an association rule of the form $\mathbf{X} \rightarrow \mathbf{Y}$ is a relationship between the two itemsets $\mathbf{X}$ and $\mathbf{Y}$, such that $\mathbf{X} \subset \mathbf{I}$, $\mathbf{Y} \subset \mathbf{I}$, $\mathbf{X} \cap \mathbf{Y} = \varnothing$ ($\mathbf{X}$ and $\mathbf{Y}$ are *disjoint*) and $\mathbf{X} \cup \mathbf{Y} \subseteq \mathbf{I}$. Moreover, an association rule is described in terms of support and confidence. The support of an itemset $\mathbf{X}$ is the fraction of transactions that contain the itemset as given below.

$$Supp(\mathbf{X}) = \frac{\text{Number of Transactions that contain X}}{\text{Total Number of Transactions}}$$

An itemset is called large, if its support exceeds a given threshold sup_{min}. The confidence of a rule $\mathbf{X} \rightarrow \mathbf{Y}$ is the fraction of transactions containing $\mathbf{X}$ that also contain $\mathbf{Y}$:

$$Conf(\mathbf{X} \rightarrow \mathbf{Y}) = \frac{Supp(X \cup Y)}{Supp(X)}$$

The association rule $\mathbf{X} \rightarrow \mathbf{Y}$ holds, if $\mathbf{X} \cup \mathbf{Y}$ is large and the confidence of the rule exceeds a given threshold $conf_{min}$. The actual process of associative rule mining is carried out in two steps: (a) all large itemsets appearing in the transaction database are determined, and (b) for

each large itemset, say **Z**, appropriate complementary subsets **X** and **Y** of **Z** are found such that the rule **X** $\rightarrow$**Y** exceeds $conf_{min}$.

4.2 Applying Associative Rule Mining to A-FHT

Here, we first introduce the notion of user sessions in the context of associative rule mining. Each user session **S** is viewed as a collection of network requests made during the time interval between system start and system shutdown.

Algorithm 1: Algo-Find-Honeytoken (A-FHT)

C_k : Candidate itemset of size k
L_k : frequent itemset of size k
function A-FHT(**I**, **S**, sup_{min}, $conf_{min}$, k_{max})
1. $k := 1$
2. L_1 = {frequent itemset of size *1*}
3. **foreach** ($k = 1$; $L_k \neq \Phi$; $++k$) do
4. C_{k+1} = Candidate sets generated from L_k
5. **foreach** transaction ***s*** in **S** do
6. increment the counts of all candidates in C_{k+1} that are contained in ***s***
7. L_{k+1} is the candidates in C_{k+1} whose support at least exceeds sup_{min}
8. **foreach** ($k = 1$; $L_k \neq \Phi$; $++k$) do
9. **forevery** non-empty set f of L_k
10. output rule f $\rightarrow$ (L_k - f) if confidence exceeds con_{min}

Figure 2: A-FHT using Apriori Algorithm to infer honeytokens

Each user session is considered as a transaction and the union of all observed network requests constitutes the itemset **I**. With specified support and confidence thresholds, *SpyZen* extracts all frequent itemsets and generates associative rules to hypothesize about the honeytoken sequence. It is important to note that it might be possible for *SpyZen* to wrongly consider frequently occurring user requests $\mathbf{N}^{\mathbf{U}}$ for honeytokens. However, in such cases, without loss of generality, we can assume that *SpyZen* can choose to either remain passive or leak out the requests by piggybacking with the information gathered during other innocuous intervals. Apriori algorithm [4], which is a classic algorithm available in the literature to learn and extract association rules, is adopted, here, in algorithm A-FHT to show how static honeytokens can be extracted from the generated network requests as shown in Figure 2.

5. Design of *SPYCON*

In order to detect SpyZen, it is essential to change the honeytoken sequence after each trial so that it is not possible for the spyware programs to deduce them. To this extent, we propose a simple honeytoken generation technique using random web spidering [27] to dynamically generate a different set of network requests each time.

5.1 Randomized Honeytoken Generation

The efficacy of the randomized *SpyCon* technique relies on the fact that a different set of network requests is generated each time to act as dynamic honeytokens; as the network requests vary it is impossible for *SpyZen* to learn the honeytoken sequence. *SpyCon* uses a technique similar to randomized web spidering is described in Figure 3.

Algorithm 2: Randomized-HT

Let X be the seed web page and G the PRN
function randomized-HT(X, G, S_{thresh})
1. **foreach** web page $n \in$ X do
2. $k :=$ G(0,1) /* randomly choose 0 or 1 */
3. **if** (k = = 1) then visit the web page n; else not.
4. **foreach** web page $n \in$ X do
5. If number_of_hyperlinks(n) > S_{thresh} then
6. X := n
7. **break**;
8. **return**

Figure 3. Spidering based Randomized Honeytoken Generation

Some web page, rich in hyperlinks is chosen as a starting point (seed), and then the hyperlinks in a subset present in the seed web page are randomly visited to generate a completely different set of network requests. The random seed and the pseudo-random sequence to visit the hyperlinks are known only to the local agent and the NIDS. A window of previously visited web pages is maintained in the event that the spidering leads to dead end with no further hyperlinks.

6. Experimentation

Towards applying the A-FHT algorithm, we conducted the experimentation in two different stages. In the first stage, we collated a series of user initiated network access patterns. This was obtained from the history logs of the users' browser. Specifically, four us-

ers contributed their history sessions. Browsing history was collected over a period of 10 days.

In the second stage, the honeytoken sequence was inserted in each days browsing history. We inserted the two types of honeytokens, viz. a deterministic/repetitive honeytoken and the randomized honeytoken (as defined in Figure 3). The deterministic honeytoken was determined by inserting links to which none of the four users had visited. Each honeytoken sequence was inserted at a random timeframe for each day of the users' history. Thus the sequence $\mathbf{N^r}$ (recorded access pattern) for 10 days was formed. The algorithm A-FHT was fed with these recorded network access patterns. The deterministic sequence was easily detected by the A-FHT algorithm, with a support of 100% and a confidence of 100%. The support and the confidence are a perfect 100% due to this reason: since we assume there are some periods of time every day when user activity is idle (e.g., lunch time), the deterministic sequence appears in every transaction (or user network activity). The confidence is also high, but for a very different reason. Consider the fact that an average user browses a search site (Google) everyday. In fact, this was the case for all the four users' profile. However, we recorded each network transaction in its entirety, i.e., the complete HTTP request. Hence for a search term "*search*", the network link would assume the form *http://www.google.com/search?q=search*. The exact form of this link varied for the different users dependent on the browser (Internet Explorer and Mozilla in our case) and the mannerism of launching the search. For example, IE version 6.0 users had a different string when they launched the search from the Google Toolbar. IE version 7 users used the inbuilt search bar and had the simple form described before. Thus, the exact link varied for each search, and even though certain searches were repeated, the support and confidence level did not approach even 30%. We considered only the top 5% of the rule (with 95% confidence) for determining honeytokens. On the other hand, for the random honeytoken generation, the support and confidence were nowhere near 30% confidence and support levels, which is similar to other normal user activity, and hence indistinguishable.

Although other data mining techniques can be used, associative rule mining has been utilized in this paper to illustrate the utility of data mining algorithms to detect any deterministic honeytokens. The purpose of using the data mining techniques is to bring out the inherent weakness of static honeytoken based schemes. It might also be possible that spyware programs using these techniques incur heavy operating overhead, thereby giving themselves away. However, the complexity of underlying Apriori algorithm primarily depends upon the size of the input data supplied, which in this case is proportional to the generated stream of network requests. More complex algorithms exist which can be used to infer sequential patterns in the data stream with less computational overhead.

7. Implementation Issues

Throughout this paper, the notion of user activity has been represented as a sequence of network requests. Practically, this is rarely ever the case; spyware usually consists of sophisticated monitoring tools, of which monitoring network accesses is only a small part. For example, the spyware [24] works by logging keystrokes; other spyware programs may use a combination of keystrokes, network access, etc. In light of such characteristics, it is apparent that the notion of network requests (only) as user activity is flawed and not practical, since spyware such as [24] can decide if a user is present on the system by checking the keystrokes. Thus, malware can sense "No user Activity", thereby rendering ineffective agents like [6] that rely solely on network activity as the detection parameter. However, this raises the interesting question, "What is User Activity?" from a parametric point of view.

Towards a practical implementation, we explore the nature of realistic user activity and the various avenues for emulating them. The notion of user activity must be all encompassing if new threats like *SpyZen* are to be detected. User activity must be considered in its entirety, i.e., not only processes that are executed by users (and background process activity as a result), but also other parameters such as keystroke activity, mouse movements, interactive sessions, etc. Towards emulating realistic user activity, we need to define and create parametric representations of user activity based on GUI [12] that are statistically similar [7] to real data sets of user activity. The local agent must be capable of emulating user activity in terms of these parameters. The basic requirements of the local agent still remain the same as in [6], viz. the local agent should not irrevocably change the system state or interfere with the users workflow.

In order to fulfill these requirements, the input generation technique for the local agent must have the following properties.

- *State Change:* To avoid detection by spyware that uses a state change as an indicator of real user activity, the local agent must change the state of the local system in a manner similar to that of legitimate user activity.

- *Emulation of user activity:* The local agent actions will emulate normal user activity in a statistically similar manner. This includes, but is not limited to, keyboard activity, mouse movement, networks access, etc.
- *Restoration of state change*: Before the real user is active again, the state change initiated by the local agent is reversed.

Note also that an idle user can be determined in a variety of ways. The local agent (and the spyware) may monitor the mouse and keyboard activity and declare the user to be idle when such activity ceases for a specified time period. Still other techniques involve simplistic schemes like detecting if the workstation is locked; to sophisticated ones like workflow process monitoring (like email, browsing, etc.). These issues bring about an interesting point: if the local agent (and spyware) can determine if the user is idle by simply determining if the system is locked, then the threat vector shifts completely, and so do the anti-spyware techniques. In such situations, it behooves the operating system to provide a secure means of hooking into specific system events, thereby ensuring that only *trusted* and *user approved* processes only obtain critical hooks/notifications.

Additionally, irrespective of the technique used, there is bound to be a time gap between the user leaving the terminal and the local agent/spyware detecting that the user is idle. This time gap can also be used by spyware to determine that the user is idle (and is probably the initiation point of honeytoken generation). Therefore, although anti-spyware techniques like [6] and *SpyCon* presented in this paper may be technically sound, their implementation must receive appropriate focus in order for these techniques to successfully accomplish their goals.

8. Conclusion and Future Work

In this paper, we presented a new class of spyware called *SpyZen* that is capable of defeating current state-of-the-art anti-spyware techniques. *SpyZen* operates in a surreptitious manner by blending in with legitimate user activity. It also defeats anti-spyware schemes that generate deterministic honeytokens to trick spyware into assuming legitimate user activity. *SpyZen* uses standard data mining algorithms like Associative Rule Mining to detect deterministic honeytoken generation schemes.

To counter spyware like *SpyZen*, we devised a randomized honeytoken generation scheme called *SpyCon* that addresses the inherent disadvantages of static honeytoken generation. Practical implementation of *SpyCon* must ensure that user activity is parameterized by a number of parameters, viz. keystrokes, mouse activity, network access, etc., as opposed to only network accesses. The randomized honeytoken generation scheme uses a root keyword, initially pre-shared between the local agent and the NIDS and generates network accesses (or other user activity profiles). The generation scheme also leverages on previous works that automatically generate user activity data sets that are statistically close to real user activity, thereby rendering most distinguishing techniques ineffective on a practical scale.

Our future work consists of integrating the honeytoken generating local agent with the remote NIDS within the framework of the Snort IDS [21]. This would also enable us to analyze the effectiveness of our solution by evaluating it against live spyware programs.

Acknowledgement

This research is supported in part by NSF Grant DUE-0402388.

References

[1] *Ad-Aware SE Personal Edition: www.lavasoftusa.com*, 2006.

[2] *The Metasploit Project: www.metasploit.com*, 2006.

[3] *Spybot Search & Destroy: www.spybot.info*, 2006.

[4] R. Agrawal and R. Srikant, *Fast Algorithms for Mining Association Rules in Large Databases*, *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*, 1994, pp. 487-499.

[5] K. Borders and A. Prakash, *Web Tap: Detecting Covert Web Traffic*, *Proceedings of the 11th ACM conference on Computer and Communications Security (CCS)*, ACM Press, Washington DC, USA, 2004.

[6] K. Borders, X. Zhao and A. Prakash, *Siren: Catching Evasive Malware (Short Paper)*, *Proceedings of the IEEE Symposium on Security and Privacy (S&P) - Volume 00*, IEEE Computer Society, 2006.

[7] R. Chinchani, A. Muthukrishnan, M. Chandrasekaran and S. Upadhyaya, *RACOON: Rapidly Generating User Command Data for Anomaly Detection from Customizable Template*,

Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC), Tuscon, AZ, 2004, pp. 189-202.

[8] M. Christodorescu and S. Jha, *Static Analysis of Executables to Detect Malicious Patterns*, *Proceedings of the 12th USENIX Security Symposium*, Berkeley, CA, 2003.

[9] M. Christodorescu, S. Jha, S. A. Seshia, D. Song and R. E. Bryant, *Semantics-Aware Malware Detection*, *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, IEEE Computer Society, 2005.

[10] T. Dwen-Ren, T. Wen-Pin and C. Chi-Fang, *A Hybrid Intelligent Intrusion Detection System to Recognize Novel Attacks*, *Proceedings of the 37th IEEE Annual International Carnahan Conference on Security Technology*, 2003, pp. 428-434.

[11] E. Kirda, C. Kruegel, G. Banks, G. Vigna and R. Kemmerer, *Behavior-based Spyware Detection*, *Proceedings of the 15th USENIX Security Symposium*, Vancouver, BC, Canada, 2006.

[12] A. Garg, S. Vidyaraman, S. Upadhyaya and K. Kwiat, *USim: A User Behavior Simulation Framework for Training and Testing IDSes in GUI based Systems*, *Proceedings of the 39th Annual Simulation Symposium (ANSS)*, 2006.

[13] Z. Guiling, *Applying Mining Fuzzy Association Rules to Intrusion Detection Based on Sequences of System Calls*, *Lecture Notes in Computer Science : Networking and Mobile Computing*, 2005, pp. 826-835.

[14] J. Z. Kolter and M. A. Maloof, *Learning to Detect Malicious Executables in the Wild*, *Proceedings of the 10th International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, ACM Press, Seattle, WA, USA, 2004, pp. 470-478.

[15] W. Landi, *Undecidability of Static Analysis*, ACM Letters on Programming Languages and Systems (LOPLAS), 1 (1992), pp. 323-337.

[16] C. T. Lu, A. P. Boedihardjo and P. Manalwar, *Exploiting Efficient Data Mining Techniques to Enhance Intrusion Detection Systems*, *Proceedings of the IEEE International Conference on Information Reuse and Integration*, 2005, pp. 512-517.

[17] McAfee, *Potentially Unwanted Programs: Spyware and Adware*, 2005.

[18] Microsoft, *Windows Defender: http://www.microsoft.com/athome/security/spyware/software/default.mspx*, 2005.

[19] A. Moshchuk, T. Bragin, S. D. Gribble and H. M. Levy, *A Crawler-based Study of Spyware on the Web*, *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, California, 2006.

[20] G. Ramalingam, *The Undecidability of Aliasing*, ACM Transactions on Programming Languages and Systems (TOPLAS), 16 (1994), pp. 1467-1471.

[21] M. Roesch, *Snort - The de facto Standard for Intrusion Detection/Prevention*, 2006.

[22] S. Saroiu, S. D. Gribble and H. M. Levy, *Measurement and Analysis of Spyware in a University Environment*, *In Proceedings of the 1st ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, 2004.

[23] L. Spitzner, *Honeytokens: The other Honeypot: http://www.securityfocus.com/infocus/1713*, 2003.

[24] Symantec Corporation, *Spyware.ActiveKeylog: http://www.symantec.com/smb/security_response/writeup.jsp?docid=2003-100918-2057-99*, 2003.

[25] A. Veloso and W. Meira Jr, *Rule Generation and Rule Selection Techniques for Cost-Sensitive Associative Classification*, *Proceeding of 20th Brazilian Symposium on Databases (SBBD)*, 2005, pp. 295-309.

[26] Y. M. Wang, R. Roussev, C. Verbowski, A. Johnson, M. W. Wu, Y. Huang and S. Y. Kou, *"Gatekeeper: Monitoring Auto-Start Extensibility Points (ASEPs) for Spyware Management"*, *Proceedings of USENIX Large Installation System Administration Conference (LISA 2004)*, Atlanta, Georgia, 2004.

[27] J. Young and T. Dean, *Exploiting Locality in Searching the Web*, *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI-03)*, San Francisco, CA, 2003.