Discouraging Software Piracy Using Software Aging

Markus Jakobsson[1] and Michael K. Reiter[2]

[1] RSA Labs
Bedford, MA, USA
mjakobsson@rsasecurity.com
[2] Carnegie Mellon University
Pittsburgh, PA, USA
reiter@cmu.edu

Abstract. Most people consider frequent software updates a nuisance. However, we show how this common phenomenon can be turned into a feature that protects against software piracy. We define a protocol for "drop-in" upgrades of software that renders a large class of software piracy more traceable. A novel feature of our approach is a *software aging* technique by which we force the updates to occur, or else the software becomes decreasingly useful over time.

1 Introduction

According to a recent study [3], the computer industry loses $11 billion annually to piracy, with 40 percent of all software programs pirated. Software piracy is a crime that traditional legal and technical methods to a large extent fail to prevent, much due to the low cost of the crime and the inherent impossibility of preventing data copying. Still, and as we show in this paper, there are practical methods that can be employed to discourage software piracy by making it less economically viable. The threat that we address here is that in which a pirate obtains, potentially alters, and then distributes copies of a piece of software in order to make a profit. We do *not* consider the situation in which several mutually trusting and coordinating users buy a piece of software together, although this is also a major concern to the software industry. We also do not consider piracy that is performed "for the fame of it", but focus on the problem in which pirated software is distributed for a profit.

Our protection method relies on periodic updates of software. Traditionally, and independently of our proposal, users want to have their software updated, typically in order to fix known bugs, to add security patches, to add new functionality, and to keep their software compatible with other programs. We consider how these two seemingly independent issues, i.e., piracy protection and software updates, can be addressed at the same time, giving us an improved system as a result. We also introduce a novel method, *software aging*, for increasing the dependence on updates.

T. Sander (Ed.): DRM 2001, LNCS 2320, pp. 1–12, 2002.

Our software updating methods discourage piracy by benefiting legitimate users while inflicting damage on illegitimate users. This differentiation of service creates a situation in which illegitimate users—in order to avoid that their software becomes incompatible to that of their surroundings—are forced to rely on the pirate for updates. This increases the operating costs for the pirate, and forces the pirate to keep in touch with its customers (which in turn increases the risk of discovery by authorities). At the same time, it increases the amount of trust the buyer of pirated software needs to place on the pirate (namely that he will provide updates). This decreases the value of the pirated software to its buyer, due to the risk that it may become incompatible if the updating service is not kept running. It is interesting to note that the customers of the pirate do not necessarily benefit from the pirate being successful; namely, the risks of the pirate's discovery increase with the number of customers the pirate needs to update. (This is in contrast to the typical economy of large operations.)

In addition to inflicting the above disadvantage on pirates, our methods can be used to shorten the life of *pirate* software below its current life span. Clearly, necessary and frequent updates further lowers the value of pirated software and further aggravates piracy by increasing the necessary updating frequency. (We note that this must be done with the convenience of the legitimate user in mind, as much as the inconvenience of his illegitimate counterpart, in order not to alter the perception of the software for the legitimate users.)

Altogether, we believe our techniques will infringe upon the economical viability of piracy by raising the operating costs for the pirate, and lowering the resale value of his merchandise. At the same time, the costs of maintaining the updates are kept low for the distributor, assuming that the updates are performed using an on-line protocol. This is a reasonable approach given that most computer users also are modem owners.

We note that our methods work even under the pessimistic assumption that the pirate is able to remove or alter any pieces of code and data that are used to detect or prevent piracy, such as code-embedded watermarks, verification of CPU identities, and similar. Also, in addition to defending against piracy, our suggested model makes software rentals easier to administer (simply by charging for the updates as opposed to providing these for free).

Outline: We begin by introducing our model in section 2, followed by a discussion of our general goals and methods in section 3. Then, we review related work in section 4, followed by a more precise description of our solution. One part of our method is the updating protocol we suggest in section 5; another part is our suggested methods to speed up "aging" of software (section 6). We conclude in section 7 by stating and arguing the properties of our solution.

2 Model and Requirements

We consider a model with the following participants:

- **Distributor.** The distributor sells software, keeps a list of registered users, and maintains a service for software updates for legitimate users. The goal of the distributor is to maximize his profit, and to discourage pirate versions of his software from being used.
- **Legitimate users.** Legitimate users purchase software from the distributor, and obtain updates from the same. The legitimate users want the piracy protection to be transparent as far as possible, in the sense that it should cause a minimum of negative side effects (such as delays and increases of file sizes). In other words, in terms of the features offered to the users, the legitimate users want their software to be as close as possible to the *ideal* of the software (where we use the word "ideal" as done by Plato).
- **Pirate.** The pirate obtains the software sold by the distributor, and redistributes (potentially altered) copies of the software for a charge. We make the pessimistic assumption that the pirate has access to the source code of the software he wants to redistribute, and that he is capable of altering (and compiling) this in order to remove any protection mechanisms. The pirate wants to maximize his profit and minimize the risks of discovery/prosecution.
- **Illegitimate users.** Illegitimate users obtain software from the pirate. Just like the legitimate user, the illegitimate user wants the software he uses to be as close as possible to the ideal of the software, again in terms of the functionality offered to the user. Additionally, illegitimate users want to maximize their profit (by buying software at a "piracy discount") and minimize the risks of failure. We assume that illegitimate users generally do not cooperate with one another, nor permit their pirate software copies to serve updates to other pirate copies (e.g., lest they be implicated as pirates themselves).[1] Rather, we assume that illegitimate users interact only with the pirate for the purposes of obtaining software.

3 Approach

Goals. We want to force the pirate to be responsible for regularly updating its customers' software. We consider the link between pirate and customer as the cost (in terms of risks of discovery) to the pirate, as opposed to the bandwidth between them once a connection is made. Requiring a link therefore requires either that the pirate can be contacted by its customer in some manner (preferably electronically in order for the illegitimate user not to be inconvenienced) or that the pirate can contact his customers[2] (again, preferably electronically). Here, the selection of the "meeting place" (which may be a public bulletin board) may be fixed over time and the same for all users; unique for each user and changing according to some pseudo-random pattern; or somewhere between these two extremes. In either case, we have that the need for communication increases the

[1] Experience with anonymous file sharing systems such as Gnutella suggests that this position is overwhelmingly the norm [1].

[2] We note that the latter contact method allows the pirate to move around, at the cost of forcing illegitimate users to register their location.

risks for legal action against the pirate, as it would allow "infiltrators" to discover the pirate and take action to trace him. This threat increases the risk of the pirated software to its users, as these will be made to rely on updates from a pirate that may either disappear to avoid tracing or be successfully traced and taken out of business.

Method for forcing interaction. We achieve our goals by letting the distributor supply software updates to registered, legitimate users. If an illegitimate user were to contact the distributor for an update, he would have to give a registration number. In case it is invalid, the distributor may supply a "random update", thereby efficiently corrupting the operation of the pirate software. In case it is valid, it allows the distributor to partially trace the criminal (from software distribution lists). He may also supply a random update in case he has already updated the software for the given user in this time period (i.e., another clone of it).

Therefore, a rational pirate would have to alter the portion of the software that requests updates in a way that it either contacts the pirate (which would hardwire a contract address for the pirate in every piece of pirate software sold) or that awaits an update from the pirate. If this is not done, then illegitimate users will be refused updates, which will lower the value of the pirated software to them, and therefore also the possible profit to the pirate.

Means for communication. The software distributor supplies updates by either the legitimate users contacting the distributor, or vice versa. Then, after contact has been established, some identification scheme is run, and the update is transmitted from the distributor to the user. The pirate has the same type of communication channel available. In addition, we assume that he may use (potentially anonymous) bulletin boards as a communication channel.

Limitations. In this paper we consider only programs that generate files or messages that may need to be interpreted by other instances of the same program; hereafter, all such outputs are referred to as "files". Numerous programs are of this form, include word processors, spreadsheet packages, and networked games. We do not consider software that works in perfect isolation, such as single-player games for a PC. However, our methods can be extended to any program by a hierarchical approach in which the operating system requires updates[3], and the operating system requires all programs it runs to be updated.

[3] On the surface it may appear sufficient that the operating system requires the programs it runs to be updated, but that it does not have to be updated itself. This, however, would allow a pirate to circumvent the protection by disabling the portion of the operating system that forces the programs it runs to be updated. On the other hand, in order for the attacker to avoid this from happening in a situation where the operating system has to be updated is to supply illegitimate users with operating system updates.

Conflict resolution. In the above, we have only considered what happens in the common case. There are two special cases of interest:

- **Synchronization problems.** It is not unlikely that sometimes, the connection between a user and the distributor is interrupted during a transfer. Whereas the typical approach to this in a standard setting is to execute the same sequence of steps again (with the same input and random strings), we note that such an approach is inadequate here. The reason is that this would allow a pirate to clone software that "hangs up" after having received the update, allowing another clone to claim that the connection was interrupted right *before* the last step. On the other hand, legitimate users must not be denied updates, creating an interesting new kind of fairness/synchronization problem.
- **Repentant illegitimate user.** It may be of interest to allow illegitimate users to become legitimate (by paying some fee closely corresponding to the costs of acquiring the software to begin with). By doing this, the pirate is used by the distributor much like an advertiser handing out samples that work for a limited time period. This corresponds functionally to selling the software using an on-line protocol, although it may require less information to be transferred. We will not elaborate on this scenario.

Method for forcing frequent updates. Typically, updates to software are currently done on roughly an annual basis, as there is not much need for more frequent updates, and as the cost and inconvenience of more frequent updates is substantial (using currently employed updating methods). As the success of our protection mechanism depends on the frequency with which updates are necessary, we wish to increase the frequency. We may, for example, want weekly or bi-weekly updates to be (automatically) made.

Note that it is not sufficient to force *legitimate* users to perform these updates. (In fact, it is the *illegitimate* users we want to force to make updates.) It appears that the only way to force illegitimate users to perform updates is to make these necessary for smooth operation. We consider a method in which files output by the software contain a version-dependent number that affects how the file should be interpreted when read or written. In order for software of illegitimate users to be compatible with that of legitimate users—or more specifically, for it to be able to interpret files received from legitimate users—the illegitimate software must be (roughly) as up-to-date as the legitimate user's software that created the file. To enforce this, the software contains a short secret that, together with the version number of the software, allows interpretation of files that are older or as old as the software (where age is measured in terms of version number). We propose a method achieving this goal, while smoothly allowing new software versions to interpret both old and new files.

We note that the functional changes embodied in updates must not be possible for an attacker to predict, since then the pirate could implement these updates directly and thus avoid that the illegitimate users are forced to request updates. Similarly, it must be infeasible for an isolated program (i.e., that of the

illegitimate user) to *determine* the functional updates (e.g., by observing files from properly updated pieces of software). Our solution offers these properties.

4 Related Work

Requiring registration (see, e.g., [7]) is a common method of protecting against piracy. However, this only protects software that has not been manipulated by a pirate, and therefore aims more toward preventing copying between friends than "professional piracy". Here we show that interaction allows to defend against a stronger type of adversary.

There are two classes of commercial products that work according to similar principles as our solution, as they use interaction to control piracy. One such product [5] allows registered users—but only these—to access a large repository of clipart. It is likely that it is verified that the access frequencies for each user remain at a reasonable level, thereby discouraging massive cloning of the accessing software. Another commercial service [4] first gives users a free virus detecting program, with a few free updates, after which the updating service only becomes available per subscription.

Our scheme is also to some degree related to the problem of fair exchange of signatures (see, e.g., [2]). Recall that we aim to obtain security of our scheme via software updates. However, for each updating period, only *one* update should be sent per software identifier (each corresponding to one sold software package). It is possible that the transmission is interrupted during an update; that one of the parties willfully terminates before the completion of the protocol; or that one party claims to have been disconnected, while he was not. It is impossible for the protocol participants to distinguish between accidentally dropped connections and intentionally dropped connections, and impossible to determine whether the transmission was interrupted before or after a certain transcript was received. This causes a situation resembling that in exchange of signatures (that one entity may interrupt the transmission in order to obtain some benefit). Our approach to address this problem, though, is significantly different from that used for exchange of signatures, as there is a very different adversarial model, and due to the inherent asymmetry of the desired exchange in our setting.

A similar situation to that described above can also occur during the withdrawal protocol for e-cash schemes. There, the solution is to repeat transcripts *identically* to avoid extraction of a higher number of valid coins by a cheater; in our setting, however, it is better to *change* the format of the transcripts for "re-connecting" updates. This is done in a way that disrupts independent clones from getting updates, while not complicating the re-transmission for legitimate users, whether connecting or re-connecting.

5 Updating Method

Initialization. The distributor $\mathcal{D}$ assigns an identifier to each piece of software he sells. This could be done either by incorporating this identity in the software,

or as is often done, as a paper document from which the user copies the identity at the time of installation. Identifiers are chosen by drawing (without repetition) random elements from a sparse space.

Updating Protocol. At predetermined intervals, the user $\mathcal{U}$ updates his software portfolio by sending in a list of program identifiers to the distributor $\mathcal{D}$. $\mathcal{D}$ verifies that these are valid identifiers, and that they have not been used for updates during the current time period.[4] Then, for each piece of software that is determined to correspond to a valid update request (i.e., a legitimate user), $\mathcal{D}$ sends out a correct update of this program. If the request is found not to be valid (i.e., if there is no such identifier registered, or if the maximum number of updates have already been performed for this time interval; both correspond to an illegitimate user) then $\mathcal{D}$ may either refuse an update, or may send a "random update". What constitutes a "correct update" or "random update" is determined by our software aging mechanism and thus will be described in Section 6. Intuitively, however, a correct update, once applied to the user's software, makes this software functionally current with other updated copies of the software (i.e., files produced by one can be read by the other) and partially backwards compatible with out-of-date versions (i.e., it can read files produced by out-of-date software, but out-of-date software cannot read files it produces). A "random update", once applied to the user's software, renders that software ineffective in reading any files created by other (current or out-of-date) copies of the program.

Remarks. In the above, we consider a setting with only one distributor. This trivially generalizes to any number of distributors, who may then either operate independently, or cooperate in updating user software. Also, we did not address the communication protocol. Assuming a public communication channel, we would use some form of encryption for sending the transcripts. For this, some form of symmetric encryption method may be employed. (The user identifier as used so far is a shared secret key for identification purposes, and may be augmented with a portion used for encryption.)

Conflict Resolution. If the transmission is interrupted during an update, the user has to request another update. In order for the distributor not to mistake such a repeated request for a separate request made by a clone, we suggest a variety of methods, potentially used in combination with each other:

- **Failure counter.** If both parties record in a local counter the number of failed attempts by this user, and if the user transmits this at the beginning of the update protocol, this allows the distributor to distinguish between a repeated transmission and an independent transmission by a clone. (Recall that illegitimate users are assumed to not cooperate with each other.)

[4] If the distributor grants multiple licenses to one site, then the updating may either be performed in a coordinated manner, or the distributor will allow a number of updates corresponding to the number of licenses.

- **Random nonces.** The distributor may send a random nonce to the user during his first move of a multi-move interactive protocol; this nonce has to be transmitted by the user during the next connection (or re-connection). Here, the distributor will know that the user received a nonce if the distributor receives a (potentially implicit) acknowledgement from the user. Otherwise, he will accept both the current and last nonce during the next (re-)connection.
- **Extra updates.** The distributor may allow a low number of (say five) repeated identifiers to get updates, thereby efficiently preventing against large-scale piracy still, but without introducing problems related to interrupted transmission (since it is unlikely that an update will fail more than five times).
- **Human involvement.** The distributor may require the user to call a toll-free number to "roll back" the state after a failed updating attempt. Here, the distributor may verify from whom the call is made, etc., before the roll-back is allowed. Moreover, if updates are sufficiently frequent, the inconvenience imposed on illegitimate users by this mechanism would already decrease the pirated software's value.

Pirate Tracing. For each update request that is recorded as being initiated by an illegitimate user, the distributor determines to what cluster of illegal copies the copy belongs. This is indexed by the identifier of the user software, and other available information. Similar methods are used for pirate software recovered by other means. This allows the distributor to determine (with some accuracy) the extent and source of the problem.

Avoiding Anonymous Pirates. In order to restrict the pirate to an approach in which he needs to be in direct contact with each customer for each update, it is important to limit the usefulness of bulletin boards, and in particular, anonymous bulletin boards. We note that if bulletin boards are used, the pirated software must initiate the update (as the bulletin board will not). Therefore, the software must carry with itself a description of where to look and for what, starting with what bulletin boards to search. If law enforcement or a representative of the software distributor gets access to a piece of pirate software, they can perform the same search. If an update is found on a bulletin board, the corresponding administrator can be pressured by legal means to remove the entry, and if submitted anonymously, to aid a trace.

6 Software Aging

So far, we have only been concerned with how to make updates, and not how to force the user to do these. To a certain extent, users will want to (and have to) update software due to naturally occurring changes of the same. However, since it is beneficial for us if frequent updates were necessary (as this makes life more difficult for the pirate), we have an interest in making the software *age*, i.e.,

decrease the time periods between necessary updates. Although this is contrary to the interests of users in a traditional setting, we suggest that it does not cause difficulties in our setting, where updates are made automatically, and without human user involvement.

Additionally, if we allow a certain flexibility in terms of what phase a user is in (e.g., any one out of the three most recent time periods) then we avoid synchronization problems and potential difficulties due to failure of getting updates (e.g., while traveling). If such a flexible timing is adapted (we address how to do this below) then we may also allow updates to be performed in an overlapping fashion (i.e., not all users need to update during the same short interval of time).

Aging. Our solution is for the software to encrypt all files[5] it outputs using a symmetric key common to all copies of that version of the software. Each file also is labeled with the time interval in which it was last modified (in plaintext), which indicates the key with which encryption was performed.

In order to avoid having to refresh all files when a key update is performed, and in order to avoid storing all old keys, we propose a method in which old keys can be computed from new keys (but not vice versa). More specifically, let $K_t = f(K_{t+1})$, where t denotes a time interval, $t+1$ denotes the next time interval, and f is a public one-way function that is infeasible to invert for the pirate. The distributor either has a trap-door key allowing him to invert the function f, or starts with a value K_T from which all "earlier" values down to an initial value K_0 are computed.[6] A "correct update" sent to a legitimate user at the transition from interval t to $t+1$ includes K_{t+1}, which the legitimate user uses to replace his old key. The correct update may also include patches to the software to add new features, fix newly-discovered security problems, etc. A "random update" sent to a detected illegitimate user would contain a random number in place of K_{t+1}. It may also include patches to the software that actively corrupts the software, so that it will no longer execute. Note that even if the illegitimate user detects the random update and prevents it from being applied to his software, the utility of his software will continue to degrade because it cannot read files output by later versions (and thus encrypted by K_{t+1} or some later key).

Remarks. For the software aging method we propose, any type of encryption scheme may be employed as a building block. For maximum efficiency, we propose using a symmetric cipher, such as DES [6]. We note that this is safe even if DES is not considered to be safe in a general setting. The reason is that we only need to protect against individual users from being able to decrypt messages or establish the key from seen messages. We do not need to prevent a more powerful

[5] A practical alternative is to only encrypt *portions* of the file, such as vital formatting-related portions or compression tables.

[6] Here, a value T exceeding the anticipated number of update periods is selected. For example, assuming that no piece of software has a life exceeding one hundred years, and assuming weekly updates, this corresponds to $T = 5200$.

adversary, such as the pirate himself, as if he determines the decryption key, he would have to distribute it to all his customers anyway (which is, by the cost and risk of doing this, a factor we base the security of our system on). We use the encryption scheme in a somewhat unusual way, in that we distribute *the same* encryption/decryption key to all legitimate users (allowing these to correctly interpret each other's files). Again, this is not a security flaw, even though it will be very easy for the pirate to get the keys (he may even *be* one of the legitimate users, and so, receive the key automatically). The reason for this is to force the illegitimate users to receive the key via updates from the pirate, if they cannot compute it themselves. Therefore, this unusual use of encryption does not negatively affect the security of our solution.

We further note that f does not have to remain infeasible to invert over the life of the software, but only require "too much" computational effort to invert for it to be convenient to do every time period by the illegitimate users. Should, however, an attack become known, allowing fast inversion of the function, then a new function has to be selected and employed. All software needs then to be updated to "refresh" all files of the old format, which can be performed by an intermediary version with knowledge of both the old key and the new key, and the corresponding one-way functions.

Allowing for flexible updates. In order to make new files readable to a software version that is not updated to the same version, the following method may be employed: Instead of using the most recently distributed key K_t for writing files in time interval t, the software may instead use $K_{t-\delta}$, where δ reflects the updating frequency necessary for a piece of software to be able to read new files from legitimate users. For example, we may set $\delta = 2$, allowing programs to be two updates "behind".

7 Claims

Our scheme hinders a pirate in that it forces him to frequently distribute updates to his clients. At the same time the piracy protection scheme is transparent to legitimate users (under reasonable assumptions) in that they will not be inconvenienced by the protection methods. We will now state these properties more carefully.

We assume that the pirate does not collude with honest users. If some user performs updates for another user, we consider the first user to be part of the pirate organization. Recall that δ determines the frequency of updates needed to ensure that a legitimate user's software remains compatible. Let $I_1, \ldots, I_N$ be N illegitimate users of the software in question. Let n be the number of updates the distributor allows to one and the same software clone (where we typically have $n = 1$), and finally, let c be the number of software packages the pirate has legally purchased from the distributor.

Let us assume that it is infeasible for the adversary to compute the trap-door of f (if there is one). We also assume that it is impractical for an illegitimate

user to invert f, or to determine the encryption key for a given time period from transcripts in his possession.

Claim 1: In order for the software of the illegitimate users to work without a significant degradation of functionality at any time, and given the above assumptions, the pirate needs to update at least $N - cn\delta$ of the users $I_1 \ldots I_N$, and at least every δ time intervals on average.

Correctness Argument for Claim 1:
By assumption, the pirate has distributed N copies of the (potentially altered) software. Assume that these users are not interacting with each other or with the pirate (after a first interaction in which they receive the software from the pirate). It is necessary for each user to get a software update at least every δ time intervals. For each legitimate software package sold, the distributor is willing to perform at most δn updates in this time period. Since honest users by assumption do not collaborate with the illegitimate users, the illegitimate users can get a maximum of $c\delta n$ updates for the c licenses the pirate purchased (if any). This leaves $N - c\delta n$ illegitimate users without updates for at least δ consecutive time intervals, considering only updates from the distributor.
By assumption, it is not feasible (neither for the pirate nor the illegitimate users) to invert f, and so, it is not possible to determine new keys from old ones by inverting f. Note that none of the trap-door information (if any) is made part of the transcripts (such as files communicated), and so, access to such transcripts cannot make the inversion easier. Finally, since we have assumed that it is *impractical* for the illegitimate user to determine the current key from transcripts (using standard methods for cryptanalysis in order to extract the key), we have now exhausted the number of ways in which a user can obtain an updated key. The only possibilities that remain are for these users not to get updates as often as needed, or to get updates from the pirate. □

Remark: Note that software that was updated $\delta + i$ time periods ago only fails to read files that were written in the last i time intervals – this allows us to expand on this argument in terms of the "quality of service" of the illegitimate users. Thus, a more general version of the above argument can be used to show the minimum number of licenses for a minimum quality of service. However, here, we have only considered "full" quality of service, corresponding to pirated software that works *identically* to the corresponding legitimate software.

Claim 2: The software of a legitimate user will work without significant degradation of functionality at any time as long as the user manages to connect to the distributor at least every δ time periods.

Correctness Argument for Claim 2:
Legitimate users are eligible to receive updates each time interval. We assume that the periodic interaction can be performed with a frequency of at least δ time intervals (i.e., that the legitimate user can connect with this frequency, and that the update protocol is not consistently interrupted after which no recovery action is taken). As long as they succeed in receiving updates every δ time intervals,

they will have a key that can be used to decrypt files written by legitimate users who have just updated their software, and therefore have the most recent keys available. Older files can always be read since it is possible to compute an old key from a new in polynomial time, given the chaining property of keys. □

Acknowledgments

Many thanks to Eran Gabber and Fabian Monrose for helpful discussions.

References

1. E. Adar and B. A. Huberman. Free riding on Gnutella. *First Monday*, October 2000. `http://www.firstmonday.dk/issues/issue5_10/adar/index.html`. 3
2. N. Asokan, V. Shoup, and M. Waidner. Optimistic protocols for fair exchange. In *Proc. ACM Conference on Computer and Communications Security*, pages 6–17, 1996. 6
3. The Business Software Alliance. `www.bsa.org`. 1
4. McAfee Secure Cast / Active Shield. `www.McAfee.com`. 6
5. Microsoft Clip Art Gallery Live. `cgl.microsoft.com/clipgallerylive`. 6
6. NBS FIPS Pub 46-1. Data Encryption Standard, U. S. Department of Commerce, 1988. 9
7. Sheriff Software Development Kit. `www.sheriff-software.com`. 6