Case Study: An Intrusion Investigation Featuring Spector Pro for Mac

Jody Forness & Dan Blank

October 25, 2012

The Complaint

STROZ FRIEDBERG

•Stroz Friedberg was hired to determine if the client's ex-husband had hacked her computer.

•Objectives:

-Was there spyware on her computer?

-If so, who installed it, how long had it been running, what information was collected, how often was the program accessed

This case started like many other cases… Stroz Friedberg was retained to determine if the client's ex-husband was spying on her. Specifically, she was concerned that he had access to her email, because during the divorce proceedings, he had presented e-mails that could only have come from her account. When asked if the ex-husband had the password to her webmail account, she replied, "Yes". Still, we were asked to review her computer for evidence of compromise. The client wanted to know the Who, What, Where, When, Why, and How.

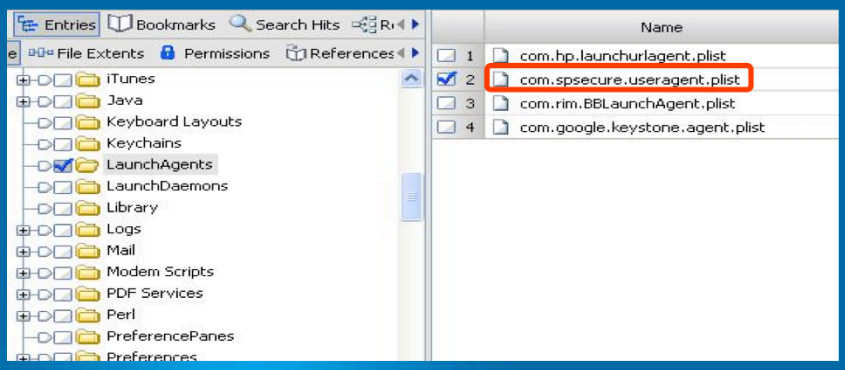
STROZ FRIEDBERG

SPECTOR PRO FOR MAC

A Suspicious Program

STROZ FRIEDBERG

- We reviewed common startup locations:
  - HDD/System/Library/StartupItems
  - HDD/Library/StartupItems
  - HDD/Library/LaunchAgents



We get these “spying” or “hacked” complaints a lot, and we have to proceed with our investigation, whether it’s probable that the client is right or not.

So, we reviewed common startup locations on the mac [CLICK], and one program stood out [CLICK]: spsecure?

spsecure?

STROZ FRIEDBERG

- "spsecure" is actually Spector Pro
- The Spector Pro application is stored in `HDD/usr/local/sps`
- The `/usr` folder is typically hidden to the user.

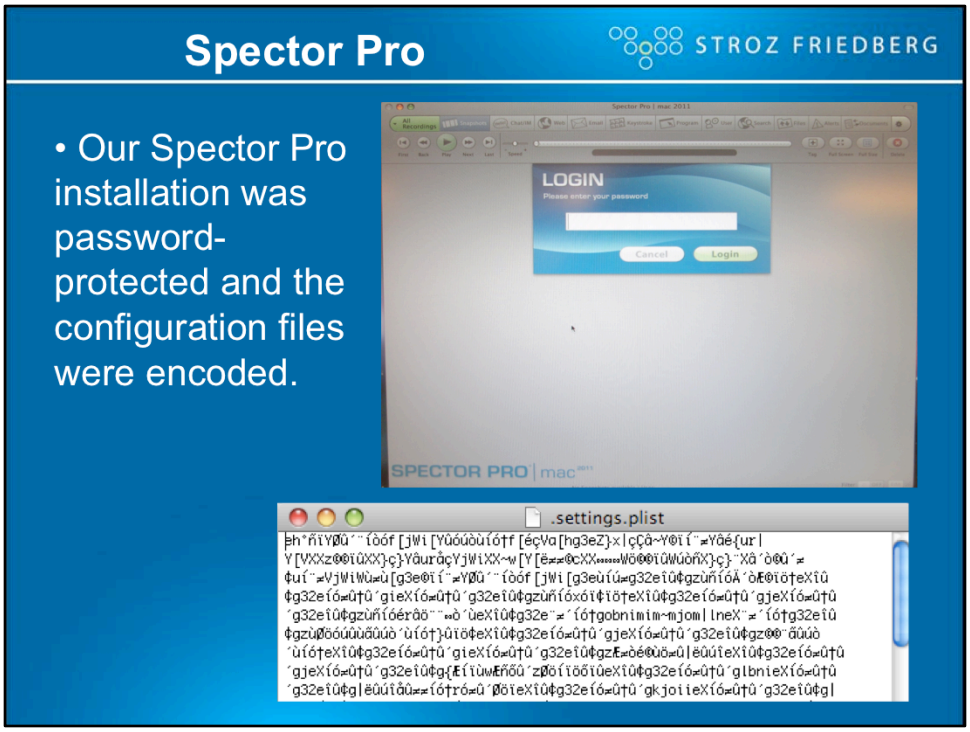
The .plist for spsecure gives the name and full path to the application [CLICK] and shows that it's configured to "RunAtLoad" [CLICK].

A quick Google searches takes you to the SpectorSoft website.

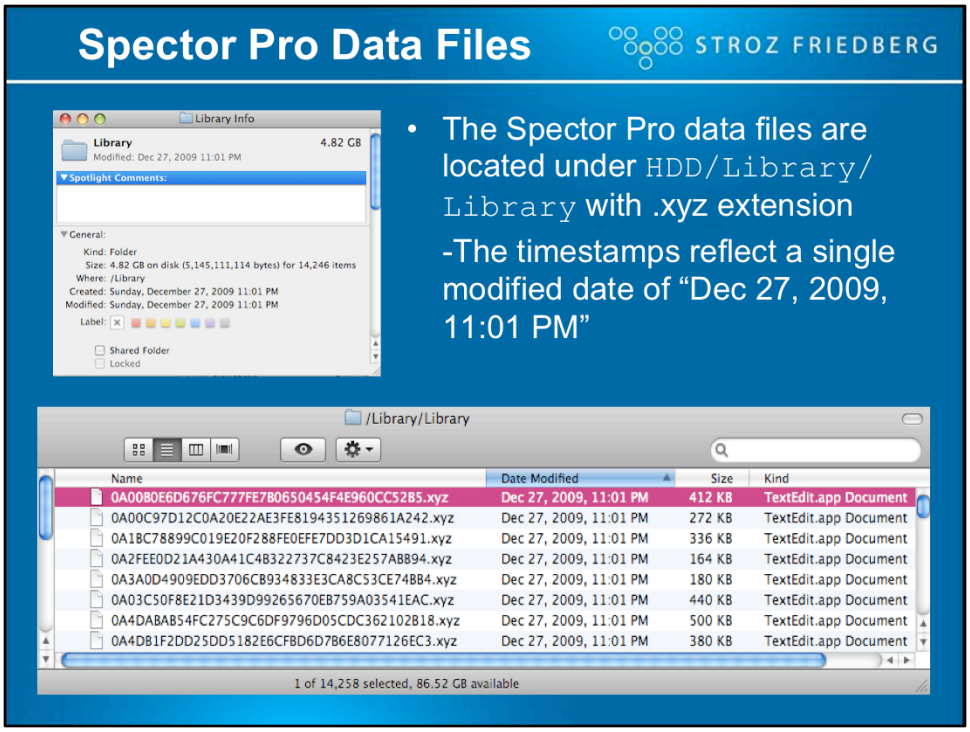
According to the SpectorSoft website, Spector Pro for mac is capable of keystroke logging, email preservation, and remote viewing, all of the activities suspected by the client. The program can be configured to run in “Stealth Mode” and is typically password-protected.

We attempted to gain access to the Spector Pro application. We knew the install location, and we were able to launch the application. In the future, we would learn that Spector Pro requires a special key combination to launch, as it does not appear with the other applications (and this key combination can be customized).

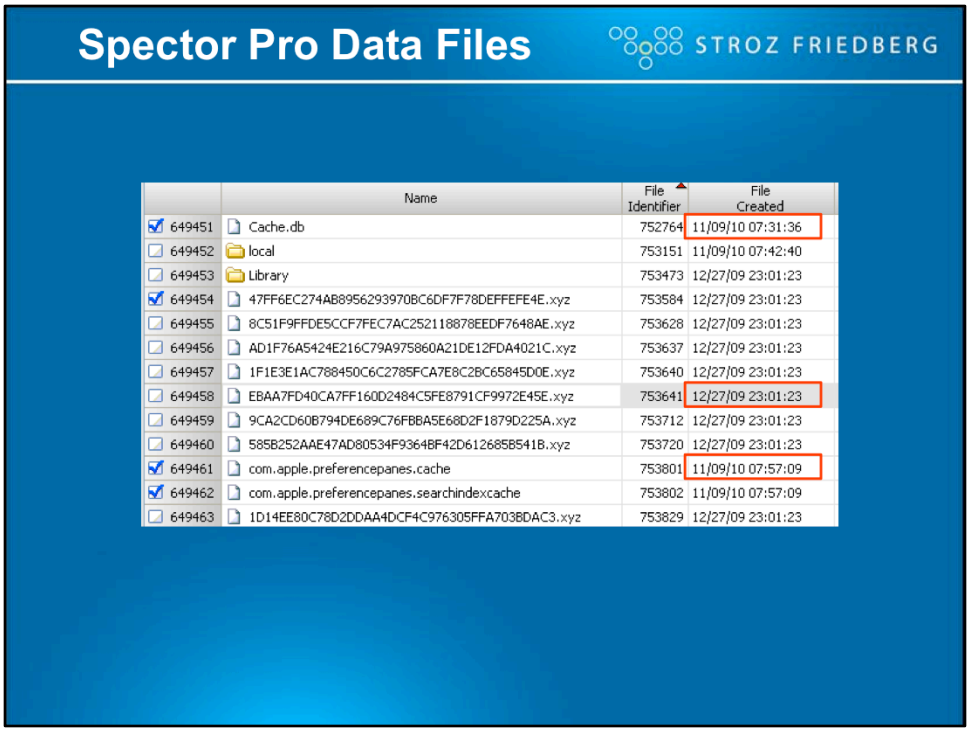
We were also able to see the config files [CLICK]: .settings.plist and .uisettings.plist (not shown here).



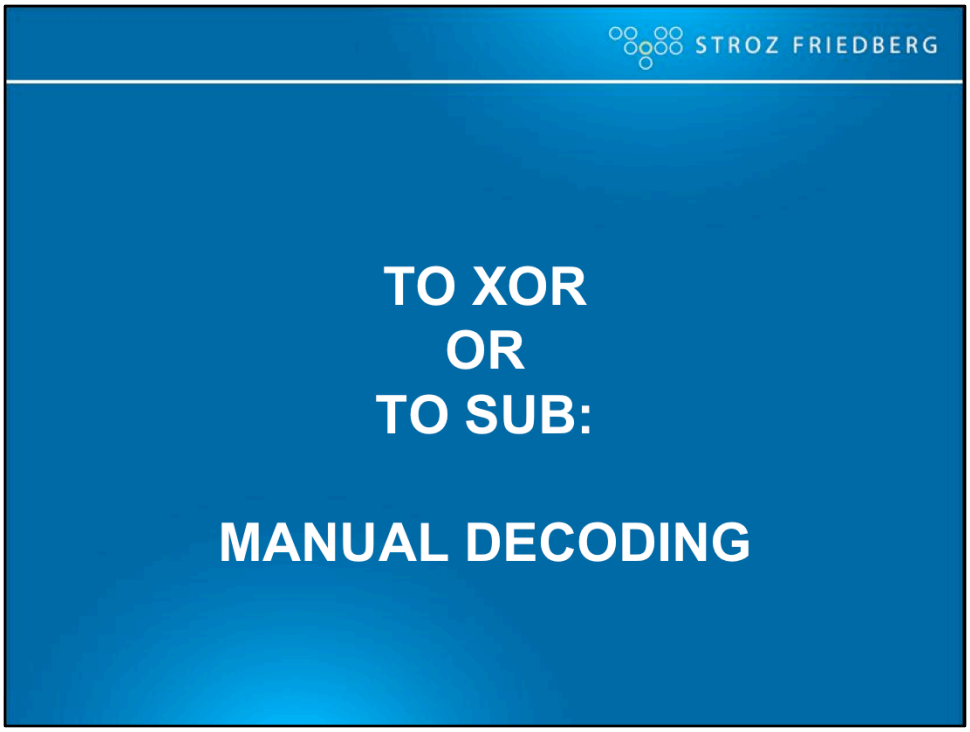
Our version of Spector Pro was password-protected and the config files were encoded. There were some log files that were in plain text, such as errorFile.plist, agent.log, recorder.log, Info.plist, InfoB.plist, RunScript.sh



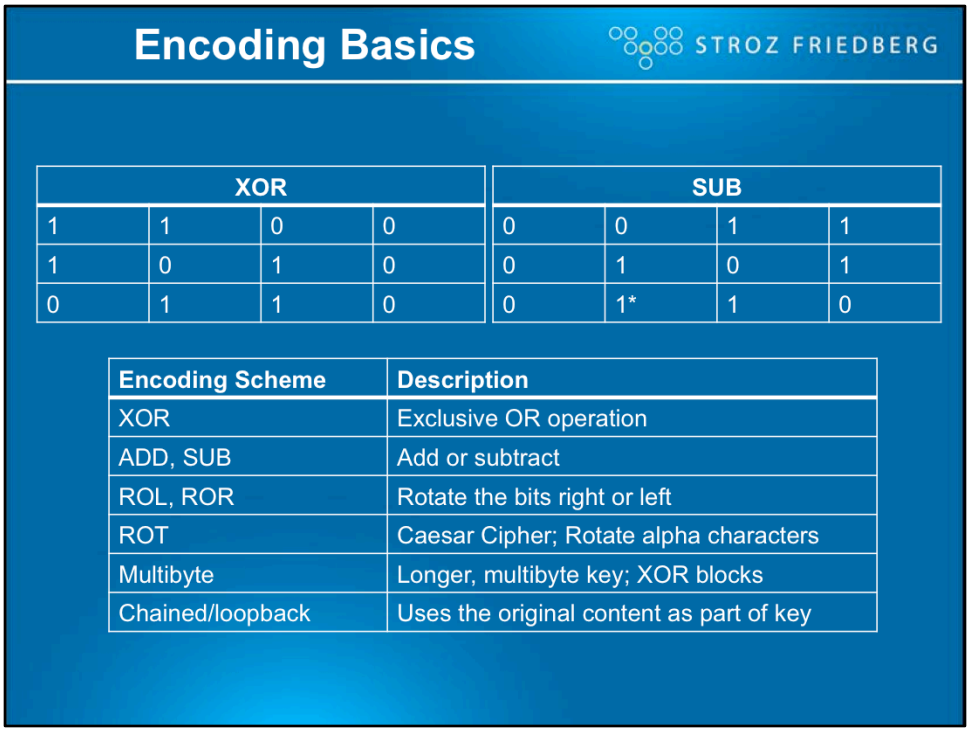
We attempted to review the files found within the Spector Pro.app folder and found little of value. Some files were in plain text, some files were encoded. One file .shared.Reg.plist,
[CLICK]
referenced the /Library/Library folder, [CLICK]
which contained thousands of files though the content appeared compressed. There were about 5GB of data total. We later learned that the screenshots were taken in black & white, possibly to keep the size of the data files small. Traditional timeline or file size analysis may not have discovered these files. The timestamps of the data files reflected a single create and modified date of "Dec 27, 2009, 11:01 PM". And the data files themselves were relatively small, considering how much data they contained.



Since these files were time stomped, we used the HFS sequential file IDs to place the files' creations in time
In this case, the highlighted recording
[CLICK]
was created between Nov. 9, 2010 07:42:40 AM,
[CLICK]
and
[CLICK] Nov. 9, 2010 07:57:09 AM.

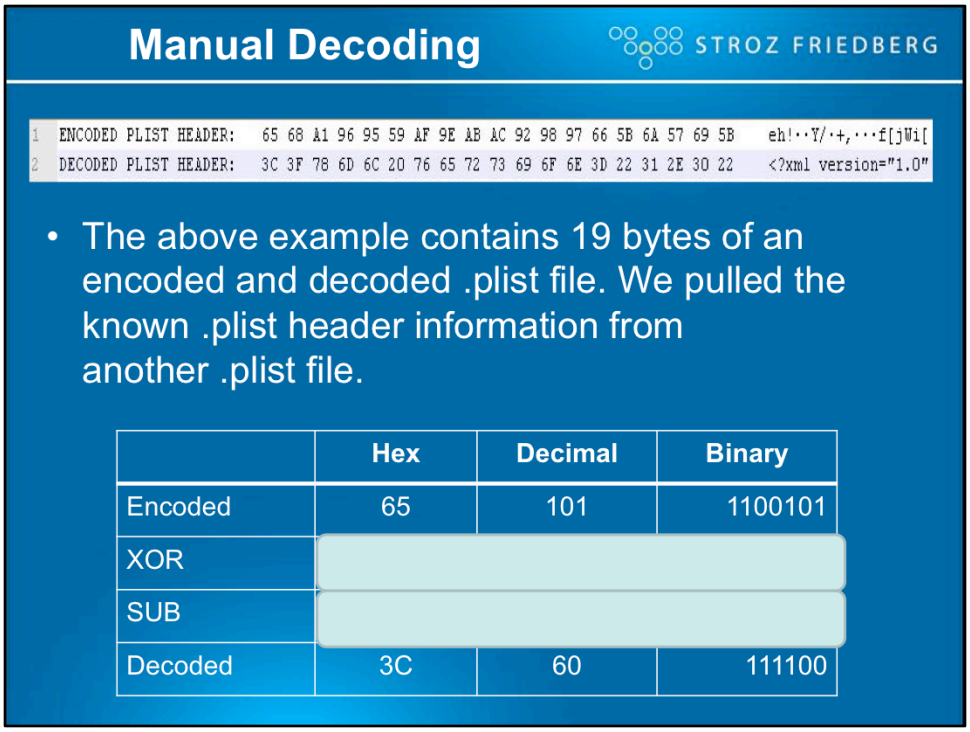


Right now, we wanted to talk about our attempts to manually decode the .plist files, because these efforts produced keywords that we used to search memory. Both of us were working on this case simultaneously, so while Jody was working on the memory dumps, Dan was attempting to decode the .plist files. In this way, we were able to work off of each other to further our investigation.



We wanted to provide the basics for typical, simple encoding schemes. This chart is straight out of a book called “Practical Malware Analysis: The Hands-On guide to Dissecting Malicious Software” by Michael Sikorski and Andrew Honig.

XOR encoding/decoding is often used in malware and malware analysis; although it’s possible to use several techniques chained together.

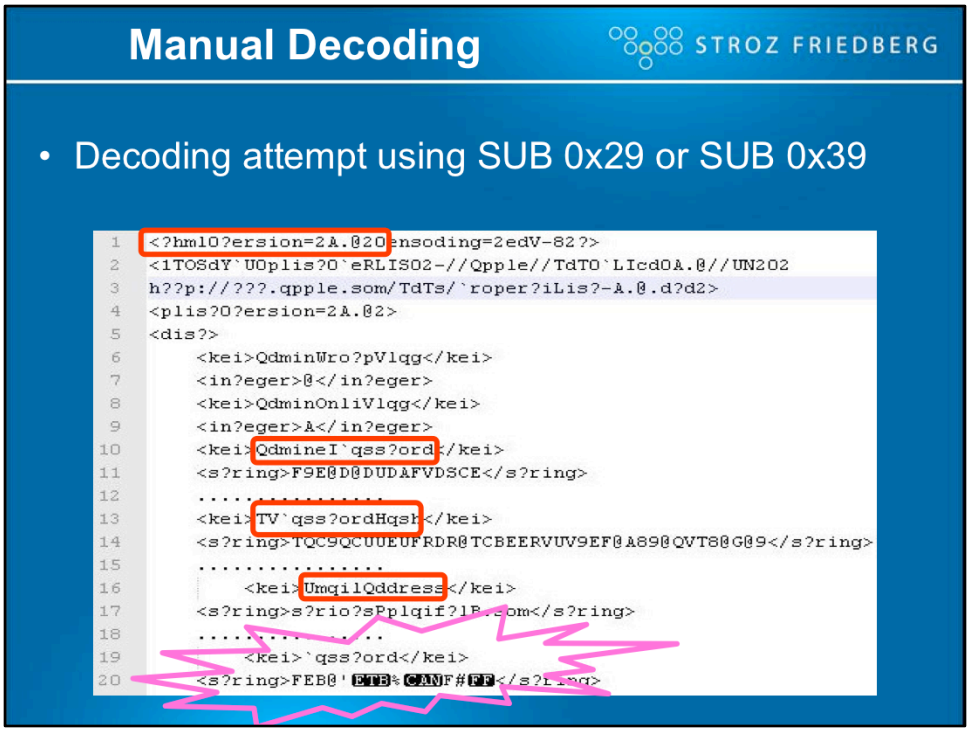


The first graphic represents the same first 19 bytes of both of our encoded .plist files and 19 bytes from a known .plist file. If the .plist file were ENCRYPTED, we would not be able to do this.

So, we have the encoded data and it's decoded equivalent (most likely). Our goal is to determine which encoding method was used and the key. The bottom table shows you the Hex/Decimal/Binary representation of the first byte.

If the data was XOR'ed [CLICK], the key COULD BE 0x59. We could test this against the next byte to see if it results in our known decoded value [CLICK].

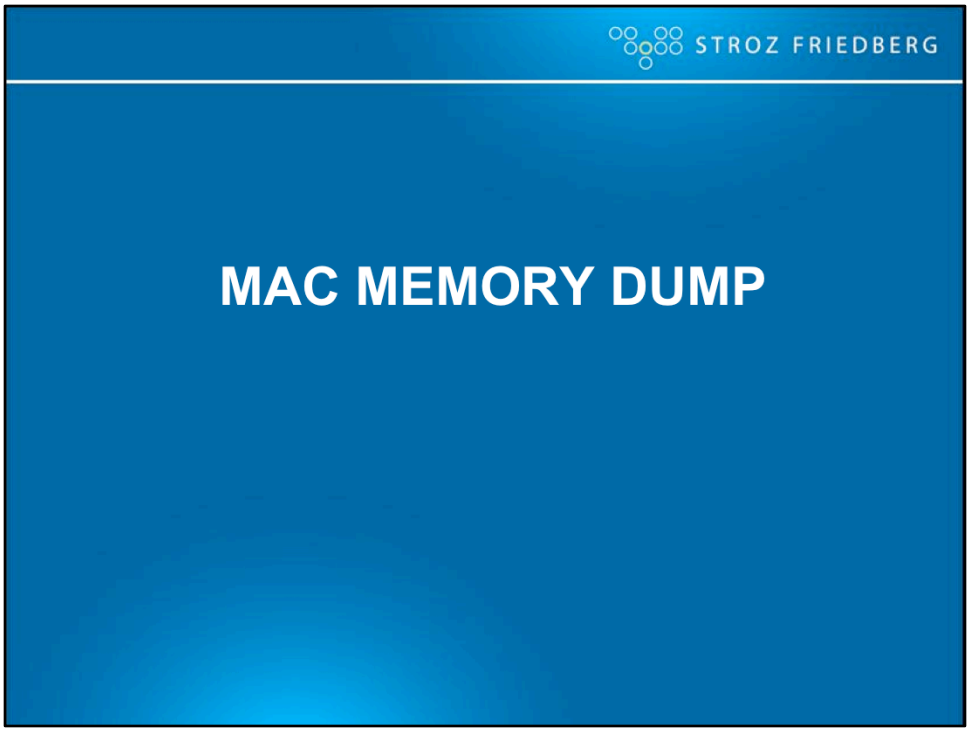
If the data was SUB'ed [CLICK], the key COULD BE 0x29. And in fact, when we test this against our known, we get successful results for the FIRST FIVE BYTES. Then, the key fails. If we do a test of the NEXT FIVE BYTES, we find that a SUB of 0x39 results in a successful decode. Then, the key fails again.



So, we attempt to decode the .plist file using SUB 0x29 or SUB 0x39 using a custom python script, and this is the result. The python script first attempted to subtract 0x29 to produce printable ASCII text. If that failed, it subtracted 0x39 from the encoded byte to produce printable text. If THAT failed, it printed a question mark and moved on. Very rough, but good enough for a first pass.

For those who are fluent in gibberish…you should be able to read this.

We see that our decoding attempt produced a partially decoded .plist file. These are notable excerpts from the complete file, but you can see the semblance of a .plist header at the top [CLICK]. Then, you can almost make out the word "AdminPassword" [CLICK]. And is that…"PasswordHash" [CLICK]. And possibly, "EmailAddress" [CLICK]. Oh my…is that "Password" with a string below it? [CLICK] IS THAT THE PASSWORD???

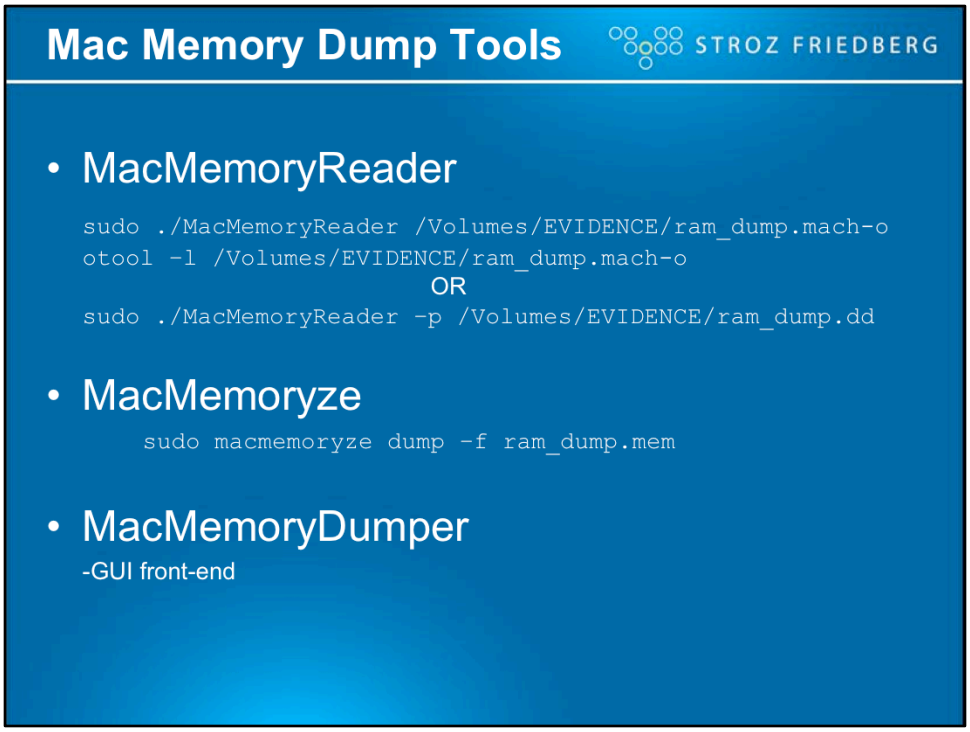


We will return to our decoding efforts in a moment.

At this point, we can confirm that the client had spying software installed, but we can't say much about its encoding. We have access to the plain text config files, and that's it.

So, we tried guessing the password to Spector Pro with no luck. The client informed us that her ex-husband used complex passwords. We even contacted SpectorSoft for password-recovery support, but they were unwilling to help us without providing the account information.

We decided to attempt a memory dump during Spector Pro's authentication, hoping that the password would be loaded into memory in plain text. We looked at three different tools: MacMemoryReader, MacMemoryze, and Inception.



MacMemoryReader can be downloaded from www.cybermarshal.com. It supports Mac OS X 10.4-10.8. The tool does require root privileges, and for this case, we did not have the user's password.

MacMemoryze is a Mandiant product. It officially supports Mac OS X 10.6 and 10.7. It also requires root privileges to run. Our custodian was running 10.5 and again, we did not have the user's password, so we did not use this utility. We've included the command-line from the README for reference.

MacMemoryDumper appears to be a lightweight GUI front-end to MacMemoryze.

Mac Memory Dump Tools

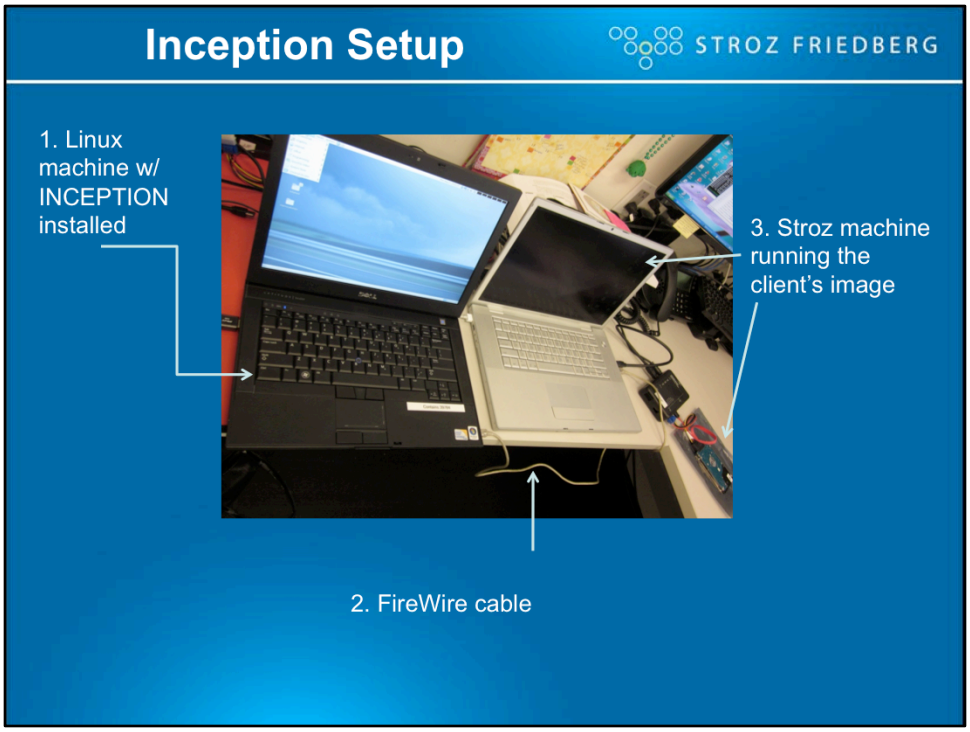
STROZ FRIEDBERG

- Inception

-Direct Memory Access (DMA) is granted to Serial Bus Protocol (SBP-2) devices over FireWire interface, and Inception utilizes this access to perform its password attack and memory dumping functions.

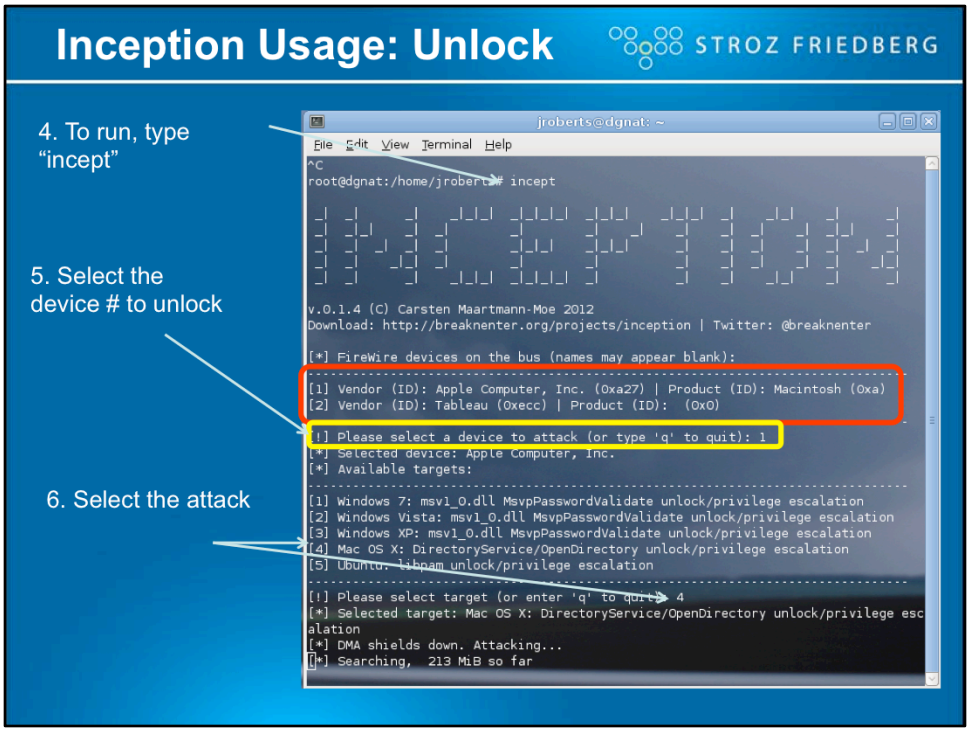
Inception is a physical memory manipulation tool that utilizes Direct Memory Access over FireWire. It can be downloaded from www.breaknenter.org. It can be used to "unlock" a computer by searching through memory for a specific offset in the OS' password authentication modules, bypassing the code that triggers from an incorrect password. Once successful, ANY password is correct.

Inception can also dump up to 4 GiB of memory over FireWire.



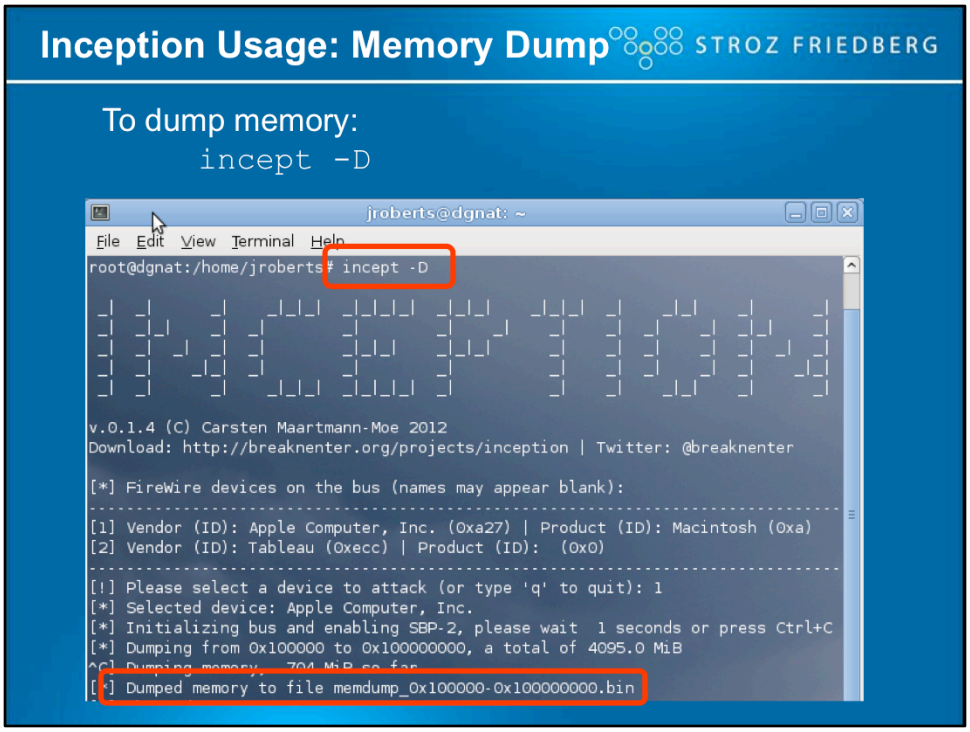
To prep for Inception, you'll need:

1. [CLICK] A Linux machine w/ Inception installed. You could use the intern's Stroz Boot CD, or we created our own installation on a large thumbdrive as part of our Inception Kit. The thumbdrive also has a large partition to temporarily store memory dumps.
2. [CLICK] A FireWire cable running from your Inception machine to a computer running the client's image.
3. [CLICK] Lab workstation booted into a blowout of the client's image.

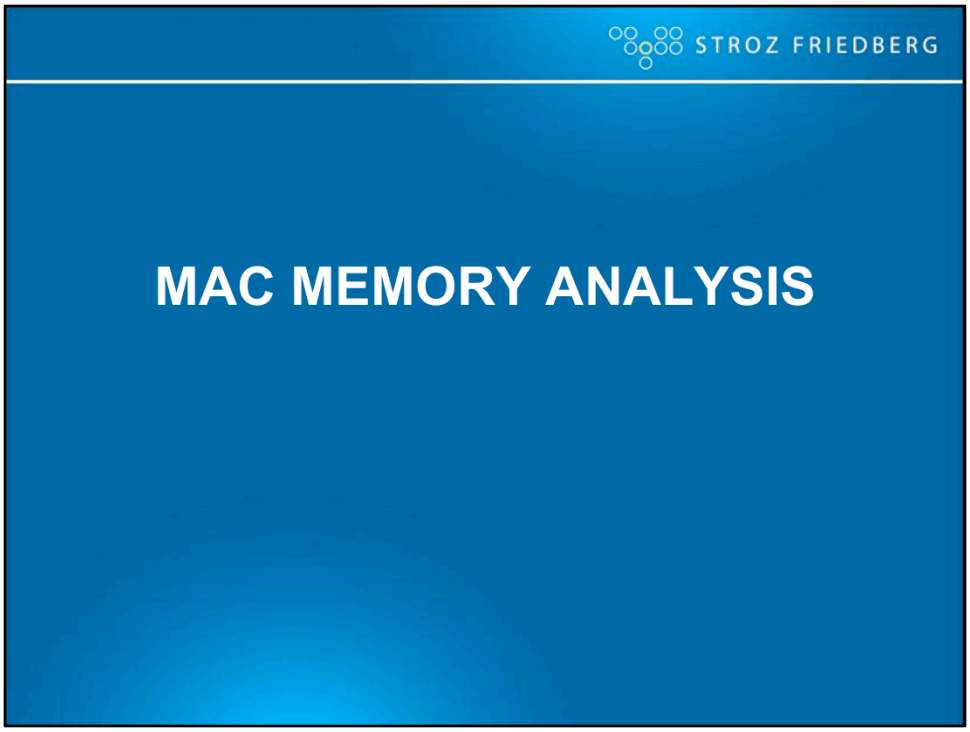


To run inception, type "incept" [CLICK].

Inception will detect all devices connected via FireWire [CLICK]. Next, select the number of the device to unlock [CLICK] and the type of attack to run [CLICK]. Inception will run and attempt to bypass the login password. This will not work on newer Macs or Macs that have been patched. You will need to be logged in first to gain DMA access.



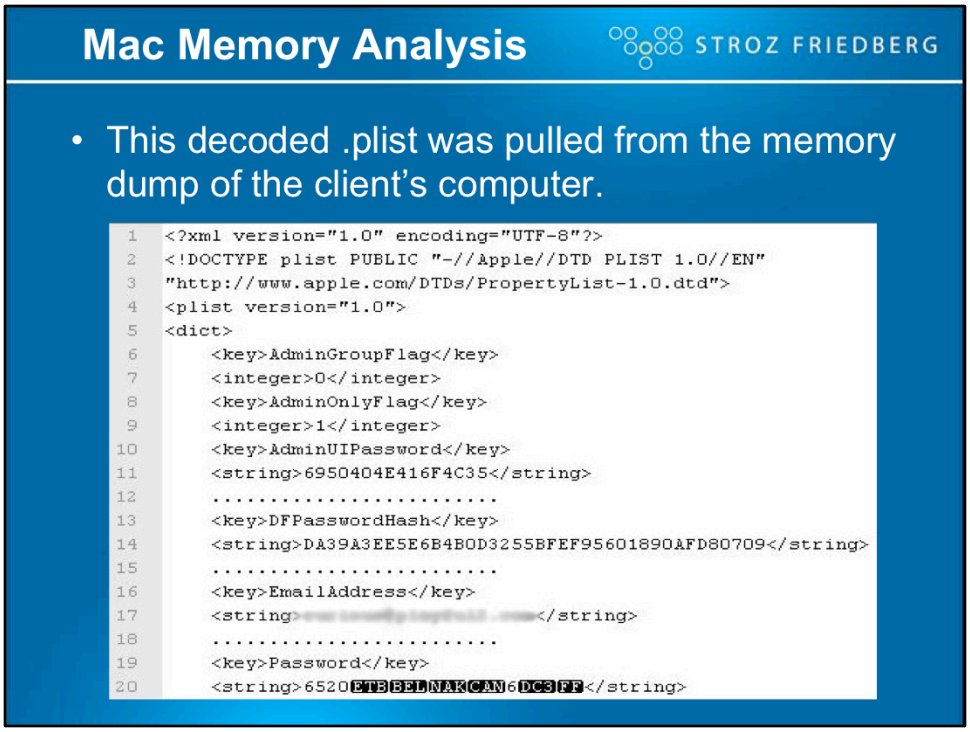
Inception can also be used to dump up to 4GB of RAM. To run, select the “-D” flag [CLICK][CLICK]. The memory dump will save to your current directory. The additional benefit of using Inception is that it’s NOT running on the target computer, meaning you’re not stepping on potential evidence by running this program. One of the only artifacts it may create is a record of a new device being attached to the target machine.



So, at this point, we have a partially decoded .plist file, but still no password.

Our brute-force exercise did provide us with some possible keywords, which we used to search our memory dumps. Between several decoding attempts, we were able to ascertain the keyword “AdminUIPassword”.

Our user was automatically logged into the machine, but we didn't have the password. Mac passwords are in plaintext in memory, among other helpful pieces of information, such as the full account name, home directory, and UID [CLICK]
A simple search expression will find them. [CLICK]
We pulled the user's password to grant us access to system settings and the ability to run other programs on the restored image.



We searched our memory dump and found the DECODED .plist file. Everything is nicely labeled and the string is readable.

For a brief second, we thought the password might actually be stored in plaintext in the .plist file; however, none of our login attempts, using these strings, were successful. What this does tell us is that SpectorPro is authenticating *against the .plist file*. So, it might be possible to replace this .plist file with a dummy .plist file that WE create with a known password.

So, we purchased a standalone license of SpectorPro for mac.

It cost $99. We installed it on a Stroz mac, which generated trusted .plist files. We attempted to copy the client's SpectorPro data files to our installation and open them that way, which did not work. The program still launched, but it did not detect the data files. Then, we tried switching our .plist file for the client's, and that WORKED. SpectorPro launched; we entered the Stroz password; and the data files loaded cleanly.

At this point, we still did not have the client's configurations, but we could see the defaults (for our version). However, we could not attribute the program, either through the registered email address or password (which we still did not have).

We turned back to memory analysis, and decided to dump memory from the *Stroz mac*, while running SpectorPro. Then, we could compare the .plist files to better understand what the program was doing.

Mac Memory Analysis

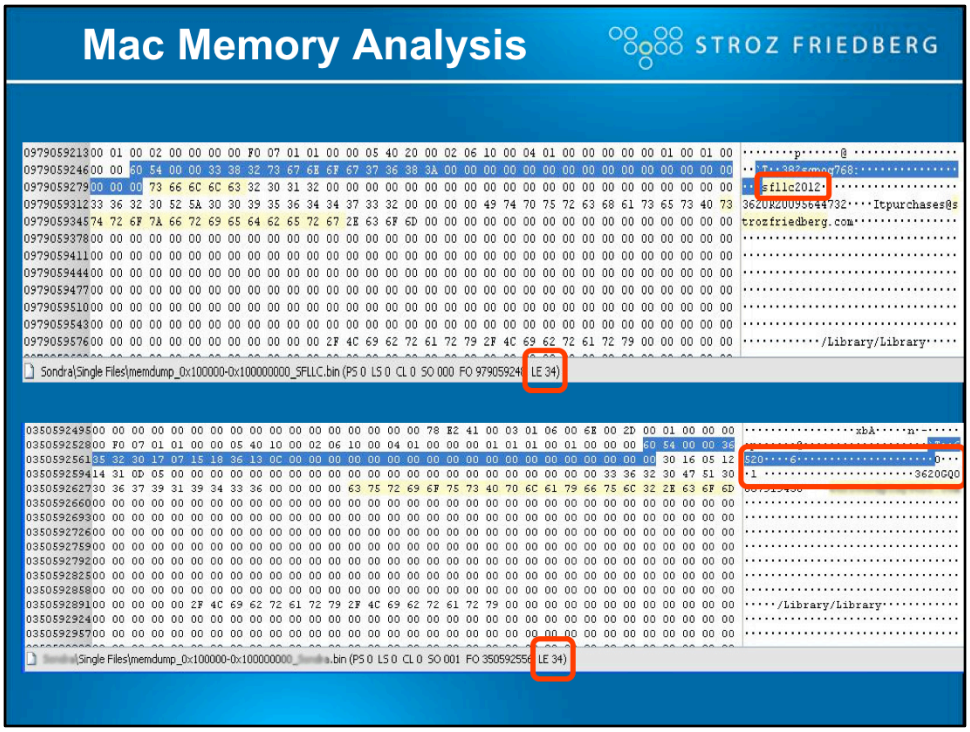
STROZ FRIEDBERG

- This decoded .plist was pulled from the memory dump of the Stroz computer.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>AdminGroupFlag</key>
    <integer>0</integer>
    <key>AdminOnlyFlag</key>
    <integer>1</integer>
    <key>AdminUIPassword</key>
    <string>ACA09798A0706F6163</string>
    ..................
    <key>DFPasswordHash</key>
    <string>DA39A3EE5E6B4B0D3255BFEF95601890AFD80709</string>
    ..................
    <key>EmailAddress</key>
    <string>Itpurchases@strozfriedberg.com</string>
    ..................
    <key>Password</key>
    <string>382sgnog768:</string>
```

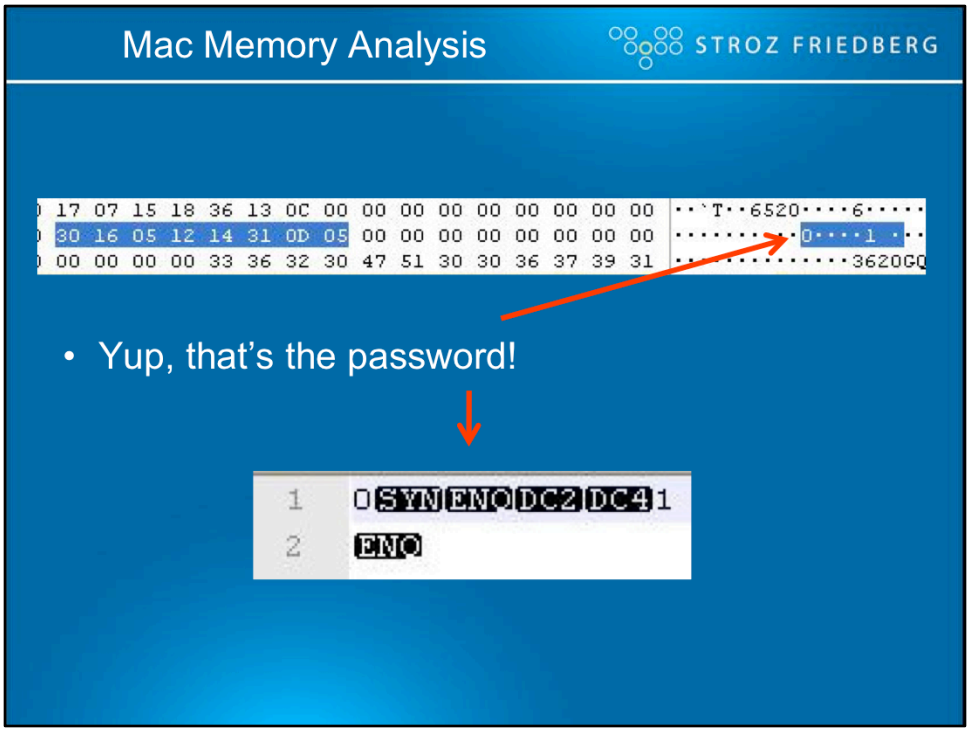
So, this is the memory dump from the lab Mac. We pulled the decoded config file for SpectorPro from memory, and we can see here that the PASSWORD IS ENCODED TOO! [CLICK]

However, now we have another keyword.



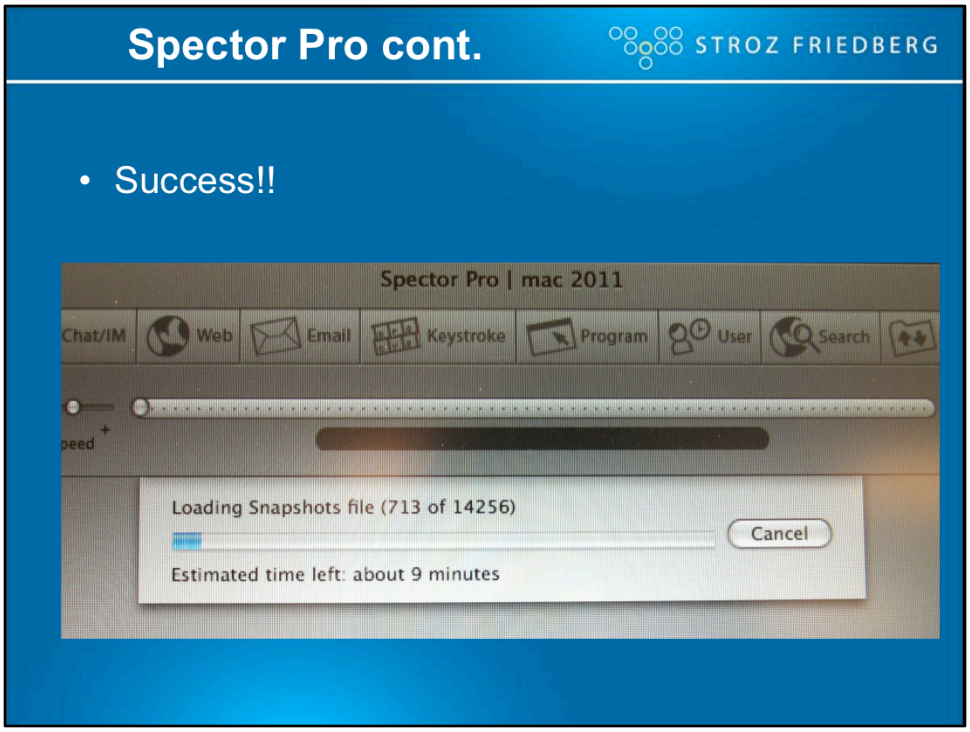
So, the first image represents a keyword search through the Stroz memory dump for the encoded password. We then highlight for length=34 [CLICK], which leads us to the password [CLICK]. As you can see, you can also search for the email address used to purchase the software.

The second image represents a keyword search through the client's memory dump for the decoded email address. We highlight for length=34 [CLICK], and we get the password [CLICK].

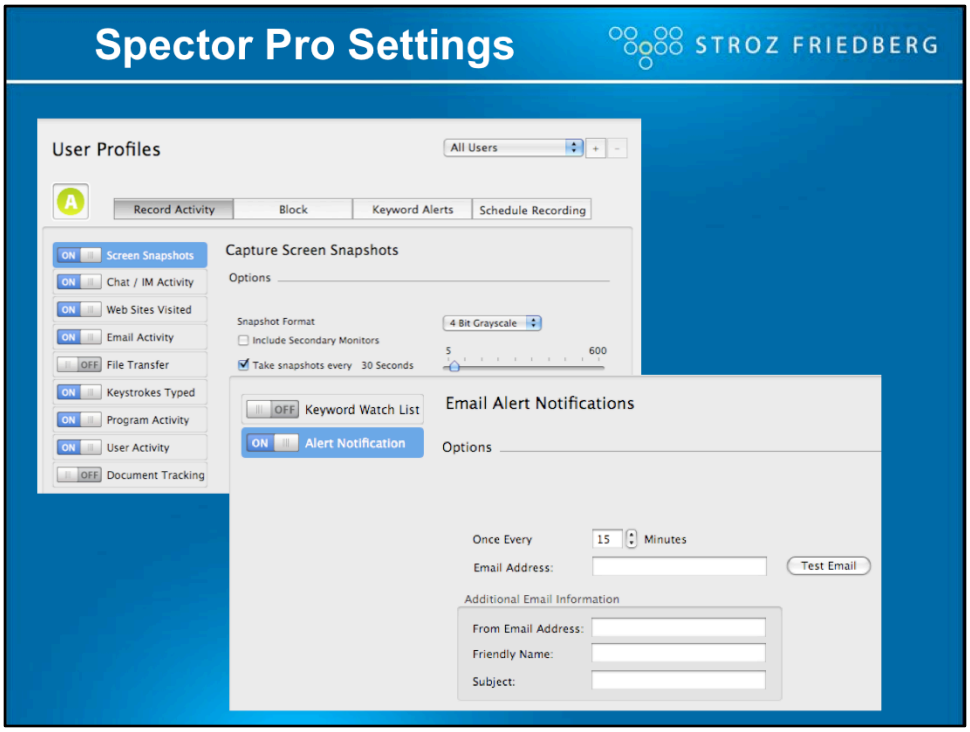


Yea, that's the password.

We did a cut/paste of the password…

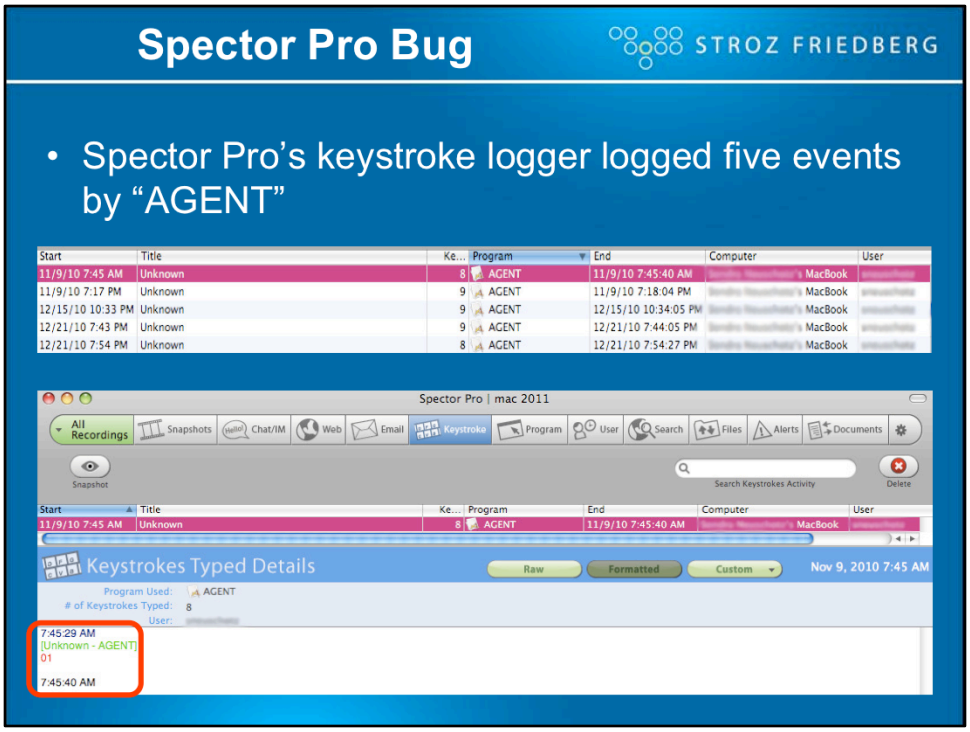


And it worked! The data files loaded, and we had access to the application!

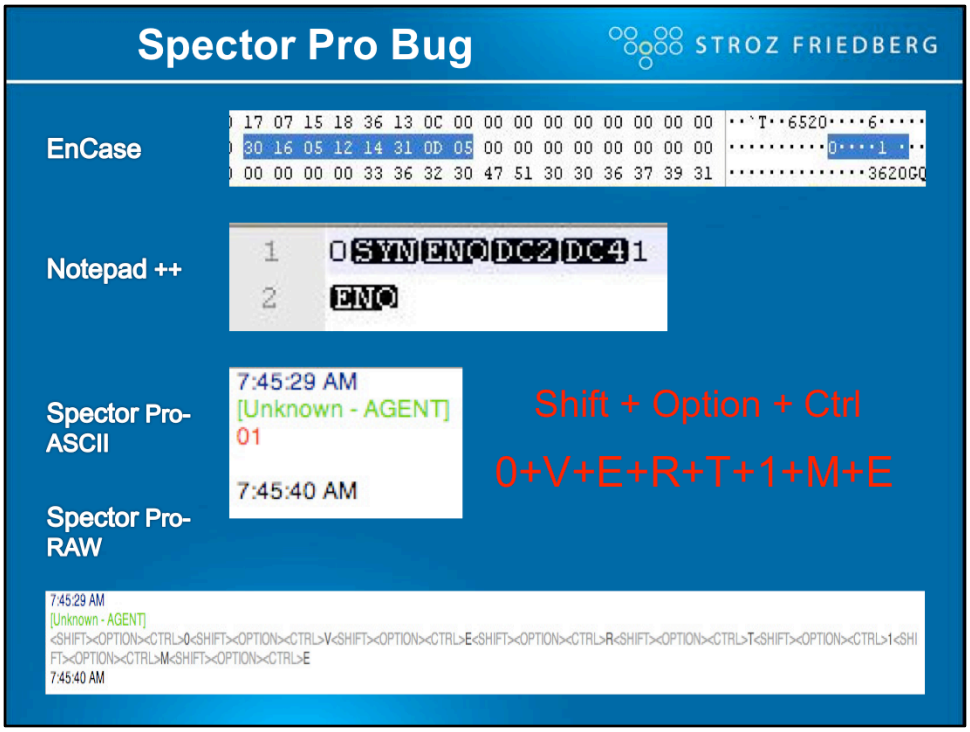


Now, we could see the settings. Spector Pro was configured to capture screen snapshots in grayscale, every 30 seconds, chat activity, web sites visited, email activity, keystrokes, program and user activity. [CLICK] Additionally, we learned that email alert notifications were NOT configured. Upon speaking to SpectorSoft technical support, they informed us that this version had limited remote capabilities, meaning the installer could configure “Email Alert Notifications” based on keywords or they could activate a share on the host workstation and configure another installation of Spector Pro on a remote computer that had LAN/WAN access to the share. This way, the user could review the data files from the separate installation.

Upon reviewing the contents of the data files, we noticed something very odd. Spector Pro had actually captured screenshots of it's OWN LOGINS and upgrades! [CLICK]



Upon reviewing the keylogged data, we noticed several unusual events which occurred early-on. Spector Pro's keystroke logger logged five events by "AGENT". [CLICK] When we look at the keylogged data [CLICK], it didn't appear to mean very much out of context.



When we finally put it into context [CLICK][CLICK][CLICK], Spector Pro had actually captured the keystrokes for it's own login!!

[CLICK] And when viewed in Raw format, these are the exact keys that were typed to login. [CLICK][CLICK]