DeSpell: An in-memory Chrome password-unlocking module

Brandon DuPree, Terrence O'Connor, Ateeq Sharfuddin

7/31/2020

Introduction

Malware is constantly evolving. Most companies today rely on virus scanners and other sensors to automate their protection. While security software will report attempts that have been made against a business, there is no way to do your own testing without running a risk to your company. For example, you cannot test the efficacy of your security rules by releasing a live malware: there is a potential to cause tremendous damage. For these scenarios, Breach and Attack Simulation (BAS) technologies were born. BAS technologies allow a company to attack their own systems to determine if the security rules and sensors are functioning correctly[1].

This paper discusses the creation of DeSpell, a module that can be deployed at run-time and executed in-memory on top of a SCYTHE client running on a computing device[2]. SCYTHE is a platform in the BAS space, and it allows you to construct threats with modules and deploy them across your enterprise network to see the efficacy of your defensive security systems. DeSpell was developed as an answer to the question: “If you had access to a user’s machine, what would you do?” Thinking from an attacker point-of-view we determined that we would want to get a list of passwords that a user has in Google Chrome since it is one of the most widely-used browsers[3].

DeSpell's purpose is to decrypt Google Chromes’ locally stored passwords. When Chrome is installed, it generates two essential files that saves data from the browser. These files are: *Local State* and *Login Data*. *Local State* stores a binary large object (BLOB) key that is used when encrypting your information, and *Login Data* is a sqlite database file that includes: usernames, website URLs, and the password field. DeSpell works by reading through a copied version of the SQL file; it then determines what version of Chrome encrypted the passwords and decrypts based on the version.

---

[1] Harvey, C., 2020. *Breach And Attack Simulation: Find Vulnerabilities Before The Bad Guys Do*. [online] Esecurityplanet.com. Available at: <https://www.esecurityplanet.com/threats/breach-and-attack-simulation.html> [Accessed 13 August 2020].
[2] Sharfuddin, A., 2020. *SCYTHE Library: Under The Hood: SCYTHE Architectural Overview (Part 1)*. [online] Scythe.io. Available at: <https://www.scythe.io/library/under-the-hood-scythe-architectural-overview-part-1> [Accessed 13 August 2020].
[3] Statista. 2020. *Most Popular Internet Browser Versions 2020* | *Statista*. [online] Available at: <https://www.statista.com/statistics/268299/most-popular-internet-browsers/> [Accessed 13 August 2020].

Related Works

Cracking passwords stored in web browsers is not a new concept. Many tools currently exist that perform this task. One of the more popular software is “Nirsoft”[4]. Additionally, Stack Overflow has many posts[5] [6] from users attempting this on their own and receiving feedback from other online members.

Most of the aforementioned projects are written in Python or C/C++ and only accomplish one decryption algorithm with the exception being Nirsoft. With Google Chrome being updated past version 80, the programs are out of date and no longer produce results for recently saved passwords. Not to mention they use external libraries that have to be installed in order to use them. DeSpell differs from these: it combines the two algorithms (AES-GCM and DPAPI[7]) that Google uses when encrypting its data along with using only native libraries included with Python.

We chose to write DeSpell as a SCYTHE module since we could run this in-memory. To simplify in-memory module development, SCYTHE provides their SDK, which allows an opportunity for others to create their own modules. You can fully customize what an attacker does to test your business's defenses[8]. We used Module Buster to generate a generic Python module.

Methodology

Creating a SCYTHE module

Module Buster allows for the creation of custom SCYTHE modules that can be run in-memory on a SCYTHE client. If you have a Python script you can create a SCYTHE module to wrap over its functionality. The process to create a module starts by selecting *New Module* in Module Buster.

---

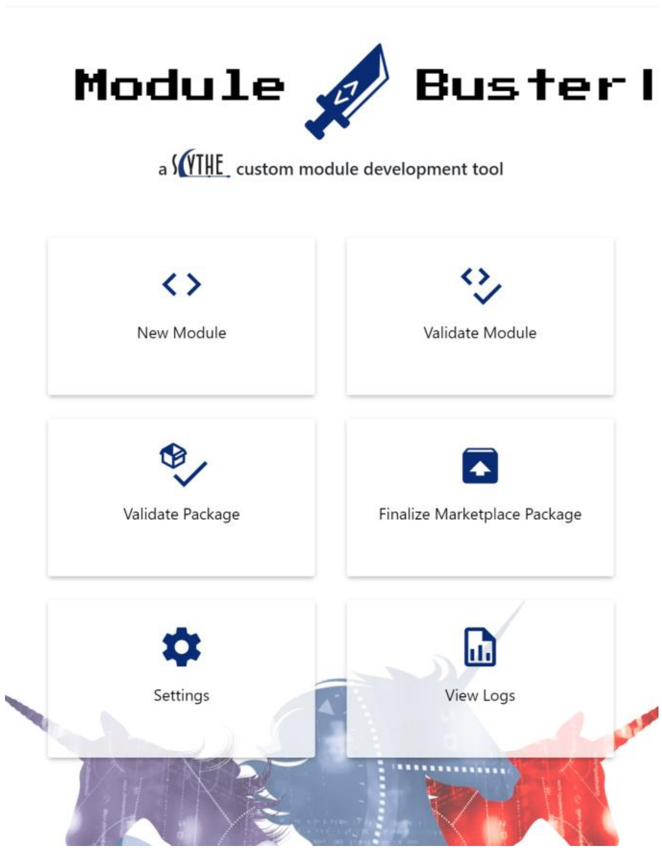
[4] NirSoft. 2020. *Freeware Utilities: Password Recovery, System Utilities, Desktop Utilities - For Windows*. [online] Available at: <http://www.nirsoft.net/> [Accessed 13 August 2020].

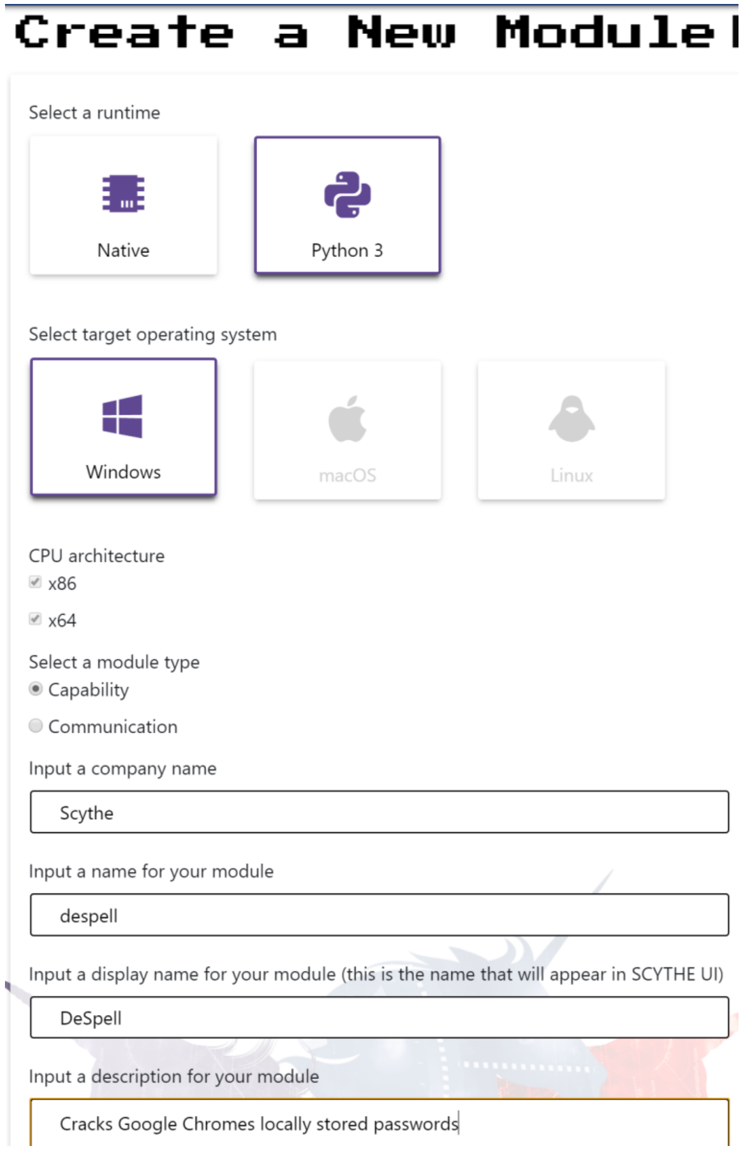
[5] Ck, R., rivers, e., Lemma, J. and Mayo, K., 2020. *I Have This Error Pywintypes.Error: (87, 'Cryptprotectdata', 'Paramètre Incorrect.') When I'm Trying To Decrypt Chrome Password In Windows*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/60329372/i-have-this-error-pywintypes-error-87- cryptprotectdata-param%C3%A8tre-incorrec> [Accessed 13 August 2020].

[6] Cristi, F., 2020. *Chrome 80 Password File Decryption In Python*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/61099492/chrome-80-password-file-decryption-in -python> [Accessed 13 August 2020].

[7] Docs.microsoft.com. 2020. *Cryptunprotectdata Function (Dpapi.H) - Win32 Apps*. [online] Available at: <https://docs.microsoft.com/en-us/windows/win32/api/dpapi/nf-dpapi-cryptunprotectdata> [Accessed 13 August 2020].

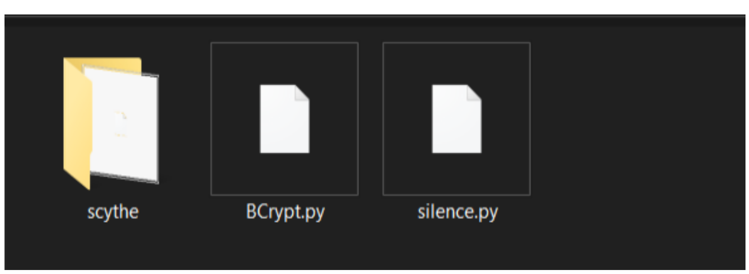
[8] GitHub. 2020. *Scythe-Io/Sdk*. [online] Available at: <https://github.com/scythe-io/sdk> [Accessed 13 August 2020].


Module Buster
a SCYTHE custom module development tool
New Module
Validate Module
Validate Package
Finalize Marketplace Package
Settings
View Logs



From here you select the corresponding runtime (for our purposes this is Python 3), then proceed to give it a name and a description along with an icon. This will generate the client-side and server-side code that you will update.

The next step we took was to copy our Python scripts and modify the client-side and server-side code. We placed our Python script to run on the client-side in the py folder. In our environment, this was `%UserProfile%\Desktop\modules\python3\dispell\windows\src\py`



We proceeded to modify auto-generated `py\scythe\despell\despell.py`. The only changes made were to import our Python script and call into it. We imported our script named silence with import silence, and then proceeded to the run method, which is called upon module invocation from the SCYTHE user interface. We added these two lines to the run method:

```
def run(message,  **kwargs):
    """
    :param bytes message:
    :param kwargs:
    :return bytes or None: None if post will happen asynchronously
    """
    # Get result of main's return
    result = silence.main()
    message = result.encode('utf-8')
    return message
```

This allows the client-side script to call our method and return the results. The server-side script was also modified to parse the results returned from the client-side script. The server-side script is located in: `src\artifacts\scripts\scythe\despell\despell.py`. Here we imported the json package since the results are being sent as a byte string within `create_message_body`.

Lastly, we adjusted the parser inside the create_parser method to align it with what DeSpell accomplishes. Mainly its description of what will happen when DeSpell is executed and removing arguments. With these changes, we validated our package with Module Buster. Our module does not accept any arguments so the module command field was left empty, nor does our module have any dependency.

DLLs on Disk

The Python libraries pycryptodome, pywin32, and psutil contain .pyd files, which are Python C extensions. These .pyd files also rely on DLLs to be on disk. Since SCYTHE's Python runtime resides in memory, there is no expectation that the DLLs will be available on the computing device on which the SCYTHE client is running. We encountered difficulty with the AES GCM implementation available in pycryptodome. Even though at first glance, it appears that pycryptodome could be loaded directly in memory, since the .pyd files do not depend on any non-system DLLs, upon inspecting the code, we realized that the .pyd files were actually DLLs and not importable Python C extension .pyd files. Specifically, `PyInit` in these .pyd always fail as per `FAKE_INIT()` macro[9] [10].

[9] https://github.com/Legrandin/pycryptodome/blob/master/src/raw_ctr.c
[10] https://github.com/Legrandin/pycryptodome/blob/master/src/common.h

Our Solution

We could update the pycryptodome package to support loading only from memory but this is time-consuming. Our solution was to instead use Python's ctypes package. The ctypes package allows you to call native shared libraries from Python. We were able to call the Windows "CryptUnprotectData' directly as opposed to using the Python package which had additional DLL file dependencies[11]. Following this, we performed the same approach for AES-GCM. We identified the BCrypt API that is available on Windows[12]. We found an example of BCrypt written in C++ code that used AES-GCM[13]. We converted this C++ code to Python and retested.

Now was the time to bring it into Scythe and test it in memory. We imported the DeSpell module package to SCYTHE and started a campaign to test the module. the user can view custom modules and allows you to select other known malware from their list. After running the emulation and executing the loader commands, DeSpell successfully ran in memory. DeSpell is available to view from in GitHub[14].

Conclusion

We were able to take an idea, turn it into code, and implement it to run in-memory as a module on SCYTHE. The module successfully retrieves passwords from Google Chrome using only default-installed Python packages and system shared libraries. The use of ctypes played a vital role and allowed for calling functions in Windows DLLs directly. Further modifications of DeSpell would be the ability to choose web browsers, use different operating systems, and choice of what information you want, be it cookies, browser history or saved credit cards.

---

[11] Tsao, S., 2020. *Problem Using Ctypes With Cryptprotectdata*. [online] Ctypes-users.narkive.com. Available at: <https://ctypes-users.narkive.com/DGzO8OPP/problem-using-ctypes-with-cryptprotectdata> [Accessed 13 August 2020].

[12] https://docs.microsoft.com/en-us/windows/win32/api/bcrypt/

[13] Torenbeek, R., 2020. *How To Chain Bcryptencrypt And Bcryptdecrypt Calls Using AES In GCM Mode?*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/30720414/how-to-chain-bcryptencrypt-and-bcryptdecrypt-calls-using-aes-in-gcm-mode> [Accessed 13 August 2020].

[14] DuPree, B., 2020. *killersprout/SCYTHE-Modules*. [online] GitHub. Available at: <https://github.com/killersprout/SCYTHE-Modules> [Accessed 13 August 2020].