Hunting the Mutex

researchcenter.paloaltonetworks.com/2014/08/hunting-mutex/

Palo Alto Networks August 14, 2014

By Palo Alto Networks

August 14, 2014 at 2:15 PM

Category: Malware, Threat Advisories - Advisories, Threat Advisory/Analysis, Threat Prevention, Unit 42

Tags: Haystack, mutex, WildFire

Summary

Mutex analysis is an often overlooked and useful tool for malware author fingerprinting, family classification, and even discovery. Far from the hypothesized "huge amount of variability" in mutex names, likely hypothesized due to the seemingly random appearance of them, practical mutex usage is embarrassingly consistent. In fact, over 15% of all collected worms share a single mutex [2gvwnqjz].

This blog was sourced from the data generated by the WildFire Analytics cloud, which processes thousands of samples a day and provides insights into various characteristics and behaviors of malware worldwide. But before we get into the details, here is a quick overview of mutexes and why they exist in the first place.

Mutex Overview

The mutex is the fundamental tool for managing shared resources between multiple threads (or processes). If you think of the threads as a whole bunch of people in a meeting, all trying to talk at once, a mutex is the baton that gets passed from one person to the next so that there's only one person talking at a time. The important thing to understand is what the mutex is really protecting. In the above example, the resource being protected isn't the right to speak, as many might think, but rather the ability to listen.

Here's a more technical example. Lets say you want to update an Internet Explorer (IE) cookie file, adding a unique identifier for use later. Naively, what you need to do is read the cookie file in, add your data to what you've read, and write the file back to disk. But what if IE is running and also updating that file? The worst case, for you, is that both you and IE read the file at the same time but you write your edits first. This is because IE will completely destroy your edits when it writes its new version of the file over yours.

The solution to this problem is to use a mutex to protect the integrity of the cookie file. A process that has the mutex knows that while it holds that mutex no other process will be accessing the cookie file. It can then read, tweak, and write the file without fear of any clobbering by other processes.

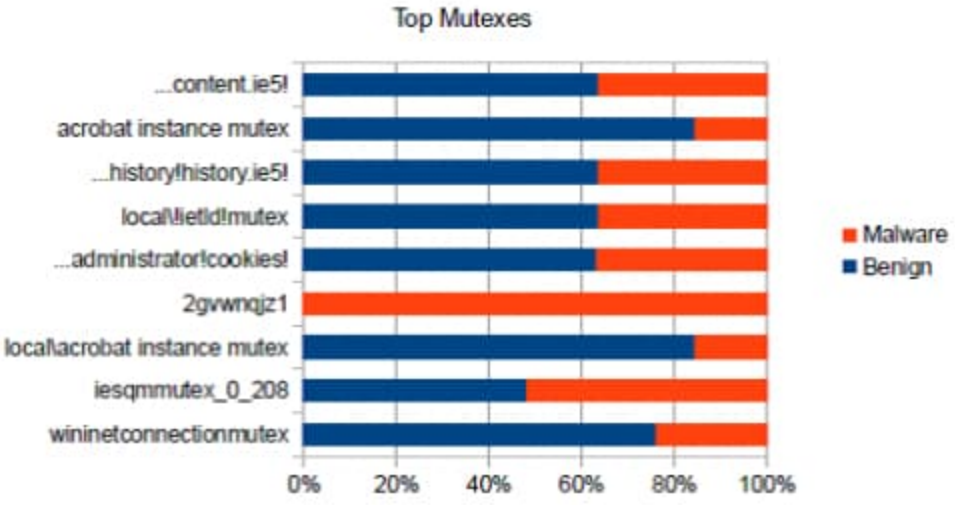
Analysis

Since each shared resource can only have a single mutex effectively protecting it, leveraging that mutex is indication that a program will be using said resource. In the cookie file example above, just referencing the mutex protecting that file indicates, with extremely high probability, that functionality to change the file exists somewhere in the program.

Any given mutex, and protected functionality, can then be thought of as an independent library of sorts. Note that while the technical implementation may not expose said functionality as a normal library, such exposure is not necessary for the types of analysis performed here. That library's usage can be analyzed in terms of who uses it. Simply put: malware writers leverage malware specific libraries and groups of like actors will reuse these core libraries when able.

Dear Haystack, You Failed

The needle in a haystack problem forever plagues malware research: it's extremely difficult to find reliable information with malware writers constantly working to undermine or eliminate that information. But, in the case of mutex analysis, the useful information pretty well slapped us in the face.



As can be clearly seen, mutex 2gvwnqjz1 is strongly associated with malware. In fact, we have only seen it in malware.

As is equally obvious, not all mutexes offer such dramatic insight. There are many common mutexes shared across both benign software and malware. What's more, they don't all share millions of uses across both sides of the fence.

In cases such as these the common approach is to use sets of the data, in this case sets of mutexes, to create fingerprints of each sample and then leverage those fingerprints to extract higher confidence classification decisions. While this avenue of research is being pursued, it suffers from all the traditional challenges of big data research. In other words, it's slow going.

In parallel, and to inform better hypothesis for the fingerprint generation, research is being done to determine how far single mutex analysis can take us. The research is ongoing but the initial results are extremely promising.

Dear Haystack, We Repent

The number of times any single mutex is used drops rapidly from the millions of samples down to thousands and from there, even further. Tens of thousands of the mutexes have been seen in only a single sample each. This results in a few hundred thousand individual mutexes available for further analysis.

What quickly becomes apparent is that a large majority of the mutexes provide no obvious means to automatically classify them as necessarily indicative of good or bad behavior. And, unfortunately, the ones, which are reasonably easy for a human to identify, are so for significantly different reasons. For example,

"autoproto_*" -- More than 20 mutexes share that preface, offering a natural fingerprint.

"global\setup_028746_mutexitem" -- Associated solely with known malware digital signers.

"defined_setnocandy" -- After reading mutex names for a few hours this just sticks out like a sore thumb.

Only the first of the examples had the mutex associated with a vast majority of malware samples. This implies that any fully automated association of a mutex to either benign or malware samples will itself require complex fingerprinting and confidence models.

Hybrid Approach

Full automation is always the ideal but it isn't always necessary. With the appropriate tools it's possible to enable a single researcher to continually review and categorize new mutexes. The initial classifications to be used are "benign", "malware", or "statistical". Meaning that the mutex either itself indicates a benign or malware sample, or that the mutex alone is not enough to make a determination and the statistical ratio of benign to malware is the best it can offer.

The backlog of already collected mutexes is too great for a small team of researchers to meaningfully tackle without some kind of ranking system. Luckily, the most objective piece of data collected about each mutex, how many samples were classified benign vs. malware, has all the information necessary to ensure that the researchers' tackle the low hanging fruit first.

Case Study: "jhdheruhfrthkgjhtjkghjk5trh"

With over hundreds of thousands of malware associations, this specific mutex is associated exclusively with the Net-worm:W32/Allaple malware family which has been around since 2006 but continues to propagate and reinvent itself through the years. Though the fact that the malware writer obviously named the mutex by rolling their face on the keyboard made it obvious before we'd done any further analysis that we'd found a unique identifier within the binaries.

This malware is well documented as a powerful polymorphic worm that encrypts itself differently every time it propagates. The evasive nature of this malware family leads to a different file hash, import hash, and only a 20% average SSDeep hash overlap between the samples. But because the mutex name is set at compile time, the mutex itself offers a common thread between all of the samples we collected and analyzed.

However, this particular mutex was associated with only a recent subset of the Allaple family.

Lessons Learned

Unlike many other avenues of research and classification, mutex name based associations provides an almost trivial means of uniquely identifying common code blocks and thereby malware families.

Case Study: "jhdgcjhasgdc09890gjasgcjhg2763876uyg3fhg"

The first thing our researchers noticed was the similarity between this mutex name and the previous one. While programmatic analysis would have a hard time associating the two, it's obvious to a human that the same face rolling technique was used to name this mutex. The author simply rolled around a bit more.

Quick follow up analysis revealed that this mutex was also associated with the Allaple malware family. More interestingly, it was another, non-overlapping, subset of the Allaple family. Several hypotheses followed directly from this observation.

1. The same author created both variants.
2. The second mutex was created at some point after the first.
3. The underlying functionality protected by the first mutex was removed or altered so significantly that a new mutex was required to provide the necessary protection.
4. This functionality is not available and/or used generally in the malware community.

5. Each mutex exists across multiple variants.

Absolute proof for a few of these hypotheses may never be realized. However, and lucky for us, the malware author was arrested in 2010 so several of the hypothesis can be verified.

The first is very likely due to the similarities present in the order of keys hit. Both begin with “jhd”. “jh” itself is more common than would be expected given that, with fingers on the home row, it requires the right index finger to move and press another key before any other key is struck. And “jh” is always followed by a key from the left side of the keyboard. These unique consistencies make it extremely improbable that two different people named these mutexes.

The second mutex appears to be a concerted effort to make the mutex seem “more random” than the first. It's immediately obvious that the author didn't move his fingers/hands much while typing the first. It’s obvious enough that the author likely noticed it when reworking this section of code. It's highly improbable that one would see the second mutex and make a concerted effort to make it appear “less random”. And, as can be quickly verified by searching through standard virus detection logs, the mutexes did in fact appear in the hypothesized order.

The third is likely due to the lack of overlap between mutex names. However, the research necessary to conclusively prove this hypothesis would be very time consuming and provide little other benefit.

The fourth is likely due to the mutexes only appearing in a single malware family. If this functionality were available in some more open source setting, and was of even moderate quality, we would expect to see it used in other malware families as well. As this functionality has not migrated outside the Allaple family, either the quality of the code is bad (see: face rolling), or it's simply not available to other malware developers.

The fifth is very likely as a change to the specific functionality these mutexes protect, with every change to any functionality, is simply not a practical method of development. And indeed, as with the second hypothesis, standard virus detection logs prove that each of these mutexes do span multiple variants of the worm.

Lessons Learned

Mutex names provide a window into the entire development process and timeline for malware. Idiosyncrasies of the malware author become apparent, the evolution can be traced, the availability or quality of code deduced, and reuse of functionality made clear with a simple mutex. No other currently used method of analysis offers such a personal view into malware development.

Conclusion

Mutex name analysis as a whole offers a unique look into the results of any sample classification system and the malware therein. While the research may never result in a fully automated decision system, it has been proven that researchers employing a hybrid approach to analysis will be able to provide critical and timely information to support the continual improvement of the classification system as a whole.

From edge case to systematic misclassifications, mutex usage is even more generally the canary in a coal mine than was previously realized.

Anecdotes

While tedious and time consuming, combing through mutex names did come with more than a few good laughs. After nearly as much debate as some of the real research, we've whittled the list down to our favorites (For the curious, these are all malware mutexes). Enjoy.

Pluguin - When penguins and wall sockets mate. Development environments don't have spell check but, maybe they should.

senna spy rock in rio 2001 virus - Subtlety. Overwhelming. There were many variations on this one, Senna's obviously proud of his work.

chinese-hacker-2 - We're not sure which is worse: that this is a legitimate signature or a sad frame job. Either way, somebody needs their computer privileges revoked.

mutexpolesskayaglush*.*svchost.comexefile\shell\open\command %1 %*@ - Putting shell code in a mutex name is right on the border of brilliant and insane. We'll leave that determination to the reader.

mr_coolface - Really, not so much, no.

don't stop me! i need some money! - http://www.monster.com/ - don't say we never did anything for you.