Unvanish: The Remarkable Peristence of Bits

Owen S. Hofmann, Christopher J. Rossbach, Brent Waters, and Emmett Witchel
Department of Computer Sciences, The University of Texas at Austin
{osh,rossbach,bwaters,witchel}@cs.utexas.edu
UTCS Technical Report TR-09-25

1 Introduction

The television show Mission Impossible began with Jim Phelps receiving instructions from a recording that subsequently self-destructs. This communication has strong privacy guarantees, but is not invulnerable to attack. The instructions can be overheard and even recorded. Instructions were often recieved in obscure locations, but it would be possible for someone other than Jim Phelps to listen to them.

The strongest guarantee of the Mission Impossible instructions is that after Jim Phelps hears his instructions, no one else will hear them. Were it to exist, a technology capable of reconstructing the instructions from the smoking remnants of the recording would jeopardize the success of tasks given to the Impossible Mission Force.

Vanish is a system that purports to provide guarantees similar to those of the Mission Impossible instructions for digital data like email, photographs, and video [7]. Users transform their data into a Vanish data object (VDO), which is encrypted with a randomly generated key whose only persistent copy is stored in a distributed hash table (DHT). The key is randomly generated from a large space, so it is difficult to guess. The user specifies an expiration time for the VDO. The DHT has a policy to delete data after 8 hours. To implement any expiration time greater than 8 hours, the Vanish system must refresh the key by reading it out of the DHT and storing it back. Because the key becomes permanently unavailable after the DHT expires it, expiration of the key should make it impossible to decrypt the VDO.

The most important guarantee provided by Vanish is that any VDO obtained after its expiration time should not be readable. This paper demonstrates a system, called *Unvanish*, that violates the Vanish guarantee. Anyone obtaining a VDO after its expiration time can decrypt it using Unvanish. Unvanish requires a constant, but modest investment in processing and storage.

The security of the Vanish system hinges crucially on one important assumption: that an attacker (without government-scale resources) will not be able to crawl or scrape the majority of the data stored on the underlying DHT. If an attacker were able to collect and store (almost) all of the data on the DHT, then data meant to "vanish" would be persistent. One could simply consult the stored DHT to decrypt a VDO after the timeout period. Assuming that scraping a DHT is prohibitively difficult is potentially problematic for two reasons.

Misaligned Incentives Vanish bases its security properties on implementation details of a system (the Vuze DHT) designed to be used differently from how Vanish uses it. We believe this is a critical mistake. Computer systems evolve to satisfy the demands of its users and maintainers, they cannot be relied on for security properties. We demonstrate that the the current Vanish system does not provide its intended security property. But even if the current Vuze environment were more favorable to the security of Vanish, the system might evolve in an unfavorable direction due to conflicting goals.

Untested Assumption The key assumption that existing DHTs are hard to scrape has been neither tested nor challenged in depth. In general, it is dangerous to base the security of deployed systems on assumptions that have not been reviewed by the security community.

The Vanish authors explicitly consider the attack we demonstrate, dismissing it as prohibitively expensive. We show that a dramatic gap exists between the cost asserted by the Vanish authors and the attack's real cost. The Vanish authors give a cursory analysis that estimates the cost to break the system to be around $860K per year using Amazon's EC2 system. In contrast, we show that we can recover a sufficient number of secret shares to decrypt almost all VDOs using only 10 small instances on Amazon's EC2 network [5]. Using reserved pricing it is possible to launch our attack for approximately $5,000 per year, or under 50 cents per hour.

Using 10 machine instances in EC2, we were able to recover approximately 92% of the key material stored by Vanish on the Vuze DHT. We directly tested our approach by creating 104 VDOs using the default settings from the Vanish website. Unvanish was able to decrypt all 104 VDOs after their retention period had expired. Decrypting

even a single VDO violates the main security guarantee of Vanish. We estimate that even using Vanish's high security settings, we could recover approximately 79% of VDOs after their retention period expires.

In Sections 2 and 3 we provide more information on the Vanish system and Vuze DHT. We describe our break in Section 4.

2 Vanish overview

This section provides an overview of the Vanish system, including its goals, its technical organization and its security model.

2.1 Vanish goals

The decreasing cost of non-volatile storage has given us the ability to store almost all communications for later retrieval. While the benefits of persistent data are numerous, it is often desirable for certain private data to be ephemeral. For instance, there have been numerous cases where access to a user's or corporation's email have been subpoenaed, and the subpoenaed party would have liked old emails to have disappeared.

At first glance achieving ephemeral data might seem easy; a user can simply delete any data that should not be retained. However, in practice scrubbing data is a difficult task for a multitude of reasons.

- Other users often store our private information as revealed in email, shared photographs, or text messages. Often, the sender of the message will want the contents discarded after the message is read. Typically, the sender lacks the means to enforce this wish against a receiver who actively wants to retain the message, e.g., by making auxiliary copies. However, even in circumstances where the receiver is agreeable to the sender's wishes, she may be unmotivated to put extra effort toward seeing that those wishes are carried out. We call this model of user behavior *trustworthy, but lazy*.
- Modern computing environments are populated by many backup, caching, and archival tools. As a result, even if a user deletes his own local copies of data, other copies may persist elsewhere, possibly without the knowledge of the user. Even encrypted archived copies are a problem as the private key might be available via legal action or other compromise. If the key ever becomes available, the user loses all privacy for the archive.
- Actively pruning data to retain only the most necessary and innocuous items can require a substantial user effort.

Geambasu, Kohno, Levy, and Levy proposed the Vanish system [7] to deal with unwanted data retention. The goal of Vanish is to expire data *automatically* after a certain user-specified timeout period. The system relies on an intriguing new technique to expire data. A user invokes the Vanish system, passing it a data object and an expiration time. Vanish encrypts the data object using a randomly generated key K. The system then uses Shamir secret sharing [10] to break the key into N shares where k of them are needed to reconstruct the key. Vanish then stores these shares in random indices on a peer-to-peer distributed hash table (DHT). The encrypted data object together with the list of random indices comprise a "Vanishing Data Object" (VDO).

A user in possession of a VDO can retrieve the data contents before the specified expiration time T by simply reading the secret shares from at least k indices in the DHT and then reconstructing the decryption key. However, if the expiration time has passed, the DHT will no longer store any share. The Vanish authors assert that the DHT policy of limiting data retention, combined with normal network churn will cause the information necessary to reconstruct the original encryption key to be permanently lost.

2.2 Technical overview

In this section we provide a high level overview of the Vanish architecture. For additional details we refer the reader to the Vanish paper [7].

Vanish relies on two principal mechanisms. The first is an *encapsulate* algorithm that takes a data object D as input, and produces a Vanishing Data Object (VDO) as ouput. The second mechanism is a *decapsulate* algorithm that accepts a as input VDO and reproduces the original data, with the caveat that decapsulation must be done within a certain timeout of T of the VDO's creation. Below, we describe each process.

Encapsulate(D) The encapsulation algorithm takes as input data D. The algorithm creates a secret key K, and then encrypts the data D under the key K to yield ciphertext C. Next, the algorithm uses Shamir secret sharing [10] to split the key K into n shares $K_1, \ldots, K_n$ where k shares are required to reconstruct the secret. Shamir secret sharing guarantees that k shares of $K_1, \ldots, K_n$ are sufficient to reconstruct K, but no information about the original key K can be recovered with fewer than k shares.

Next, the algorithm chooses an "access key" L, which is used as a seed to a pseudo random number generator (PRNG). The algorithm runs the PRNG to derive n indices $I_i, \ldots, I_n$. For $j = 1, \ldots, n$ it stores key share K_j at index I_j in the DHT. [1] Finally, the VDO V is output as the tuple $V = (C, L)$.

Decapsulate($V = (C, L)$) The decapsulation algorithm accepts a VDO $V = (C, L)$ as input. The algorithm seeds the PRNG with the access key L to retrieve n indicies $I_1, \ldots, I_n$. It then retrieves the data values from the DHT at these indices. If fewer than k values are retrieved, the algorithm outputs failure. Otherwise, it uses Shamir secret sharing on k shares to reconstruct a key K'. Finally, it attempts decryption of C using K'. The algorithm outputs a failure if the decryption is not sucessful; otherwise, it returns D, the result of the decryption.

Implementation The Vanish core system consists of a software package that can be installed on a Linux, Mac or Windows system. In core sysytem is coupled with a Firefox plug-in which allows the user to right-click on a selected area of text in the browser to transform this text into a Vanishing Data Object. The output of this process is a VDO encoded in text. A user can then right-click on the VDO and ask the plug-in to retrieve the original object.

The currently deployed Vanish system stores key shares on the Vuze [1] peer to peer distributed hash table (DHT). The default values for the Firefox system are to distribute $n = 10$ shares with a threshold of $k = 7$.

2.3 Security Model and Assumptions

The goal of Vanish is to provide a type of forward security, where past objects are secure if the VDO is compromised after its expiration time. This is somewhat similar to forward secure signatures [2,8] and forward secure encryption [3]. However, in these systems, a user's machine is responsible for evolving evolving (updating) a private key. In Vanish, the goal is to achieve security without requiring active deletion of the VDO from the user's machine. Instead, the system relies on the DHT data retention policy and peer-to-peer membership churn to expire the shares of the key used to encrypt the VDO.

Consider a user that creates a VDO V with expiration time T. If Vanish is secure then any attacker obtaining the VDO at time $T + t$, $t > 0$ will not be able to reconstruct the data. Providing this guarantee of security requires *at least* two assumptions about the attacker. (We note that if we require *correctness* we also must assume availability of the DHT before the timeout.)

Limited Network View The attacker must not be able to view the user's traffic to the P2P network. Otherwise, the attacker could simply sniff and store all shares pushed by the user onto the P2P network.

Limited View of DHT The attacker must not be able to read all data stored on the DHT. Otherwise, the attacker could simply backup all information stored on the DHT.

Our work focuses on the latter assumption.

3 Vuze background

The Vuze DHT is used by the Vuze Bittorrent client, primarily to implement decentralized tracking. The DHT is based on a modified Kademlia [9] implementation, and functions simlilarly to many other DHTs. Nodes in the network and keys in the hash table are assigned 160-bit identifiers (IDs). Each DHT node stores those keys which are closest to it in the ID space. To deal with unreliable nodes, keys are replicated to the 20 closest nodes to the key's ID.

The Vuze client categorizes peers into a number of *buckets* by their distance from its own ID. The client keeps a predetermined number of peers in each bucket. To store or look up a key, the requesting node hashes the key to obtain its ID, and contacts the k closest peers to the desired ID. Each peer returns a list of its own peers that are closest to the

[1] We note that the use of a PRNG is not strictly necessary, as the algorithm could simply select n random indices at the expense of storing larger VDOs.

desired ID. The requesting node contacts those peers, reaching successively closer peers until it finds those responsible for storing the desired ID. It then requests that those peers return or store the associated value. Stores and lookups are performed on the 20 nodes closest to the desired ID.

In the Vuze DHT, node IDs are generated by hashing the node's IP address and port number. To support the common case of multiple users communicating via a single NAT device, Vuze allows the same IP to join the network at different locations based on port number. Thus a single machine may join the network at multiple positions in the ID space by changing the port number on which it is running the Vuze protocol.

A node joins the Vuze DHT by contacting a known peer and intiating a lookup for its own ID. It uses this lookup to build its list of peers and eventually finds the nodes closest to its ID. When a node is contacted by a new peer with an ID among the 20 closest to its own, it replicates all of its stored keys to that node.

4 Breaking Vanish

Unvanish compromises the security of Vanish using a Sybil attack [4]. Because Unvanish attacks only the Vanish system, and not the Vuze network, it does not need to maintain a large number of nodes in the network to exert disproportionate influence on the operation of the DHT. Instead, Unvanish requires only a brief window of read-only access to the shares that comprise a VDO's encryption key, at any time in the 8-hour window between the VDO's creation and expiration (the default , and minimum Vanish expiration period). An Unvanish machine will gather all the information it can using one identity and then abandon that identity and start up a new Vuze identity. We found that one machine instance can concurrently support up to around 500 Vuze identities, making it possible to mount a successful attack with limited resources.

Unvanish reads keys and values from the Vuze DHT by joining the network in different locations. Unvanish creates vuze clients that quickly traverse all possible node IDs that can be generated by changing port numbers on a single machine. Each client creates a node on a port, joins the network, and waits to receive replicated keys from nearby nodes. Keys are archived to local storage for attacking VDOs at any point in the future. After a node has waited enough time to receive replicated keys, it shuts down, its resources are released, and a new one is created on the next port in sequence. A number of nodes run concurrently, to quickly traverse the available port numbers while still allowing each node sufficient time to join the network and store keys. A large number of concurrent Unvanish nodes joining and leaving the DHT in rapid succession copy and archive a significant fraction of the key space. By accessing large portions of the key space from each individual machine, Unvanish minimizes the resources required to record enough keys to decrypt most VDOs after they have expired.

Unvanish reduces storage requirements by ignoring many of the values present in the DHT. Values from the Vanish system are sufficiently long to encode a share of an encryption key, while the majority of values in the Vuze DHT are only a few bytes long and can be discarded.

If Unvanish can read and archive a large proportion of the keys in the Vuze DHT within the expiration time of a VDO, then it can decrypt that VDO by retrieving shares of the encryption key from our local archive at any point in the future, rather than by retrieving them from the DHT within their limited expiration time.

To quickly traverse the available port numbers while allowing each node sufficient time to join the network and store keys, 50 nodes run concurrently; each node remains active for approximately 150 seconds. Unvanish is limited to a small number of nodes on a single client due to network, processor and memory constraints of the small EC2 node and the relatively unoptimized Vuze DHT source code.

EC2 implementation We implemented Unvanish using Amazon's EC2 system, enabling a realistic assesment of the actual cost to run Unvanish. We ran our Vuze DHT client on 10 of Amazon's "small" EC2 instances, which provide 1.7GB of physical memory, 160GB of local storage, and compute power approximately equivalent to a 1.0Ghz Xeon. Memory and processor constraints restrict Unvanish to 50 concurrent DHT nodes on each instance. Each DHT node remains on the network for 150 seconds before being replaced by a new node.

Running the EC2 "small" instance costs the user $0.10 per hour if the instance is created on-demand. Thus, the cost to run Unvanish is $1 per hour, or less than $9,000 per year. By contrast, the original Vanish paper claims such an attack would have a price tag of $800,000 per year. Moreover, we believe that the cost of running Unvanish can be further reduced. Amazon provides reserved instances, whose priciing structure entails an upfront charge followed by a reduced per-hour usage rate. A 1-year reservation for 10 instances running full-time would cost less than $5,000.

	Placed in DHT	Recovered
VDOs	104	104
Key shares	1040	957

Table 1: Unvanish results for 104 VDOs placed in the Vuze DHT over a 7.5 hour window. At 7/10 shares required to decapsulate a VDO, all VDOs were successfully recovered. 957/1040 key shares were recovered.

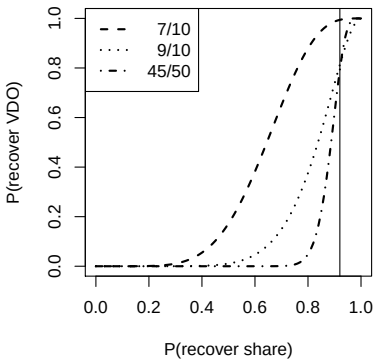

Figure 1: Estimated probability of recovering a VDO given the probability of recovering any one key share. The 92% probability of recovering a key share in our experiment is indicated with a vertical line. At that probability, we estimate that we can decapsulate 79% of all VDOs even at the recommended "high-security" setting of 45/50 shares.

Results We ran our Vuze DHT client on 10 Amazon EC2 instances for approximately 24 hours. Over a 7.5-hour window during that time, we seeded 104 VDOs into the DHT, using the default security parameters of 7 of 10 shares required for decryption. Table 1 shows the results of our experiment. Out of 1040 key shares, we were able to recover 957. We successfully decrypted all 104 VDOs using only our offline archive of the Vuze DHT.

Analysis The most immediate conclusion from our work is that the currently deployed Vanish system does not expire information when faced with an attack of modest cost. Figure 1 estimates the probability of recovering a VDO given the probability of recovering any individual key share. We model the probability of recovering a key share as a binomial variable, and estimate the probability of recovering the VDO as $P(VDO) = P(X \geq threshold)$ for n binomial trials with probability p, where n is the total number of key shares and p is the probability of recovering a share. Even at the recommended "high-security" setting of 45/50 shares, we estimate that we can recover 79% of all VDOs.

The Vanish authors consider and dismiss a Sybil attack. Here we review the arguments. The primary argument is that Vuze implements rudimentary defenses against a single node using multiple ports to gain disproportionate influence in the network. Vuze allows only a small number of stores to a given key from a single IP address, regardless of port number. Future versions of the Vuze DHT protocol will restrict the number of IDs attainable from a single machine by computing the port number modulo 1999 before hashing. Contrary to statements in the Vanish paper [7], this defense is not currently active. Even if the defense were made active, it appears that compatibility for older protocol versions will be maintained, with the result that clients can user older versions of the protocol to circumvent the defense. . The problem is basing the security of a system on the properties of different system intended for a different purpose.

The worldwide scale of Vuze is essential to the security of Vanish, and the defense needed by Vanish to thwart our Sybil attack is not aligned with the priorities of Vuze's main user base. A large DHT with nodes run by independent people with significant churn is essential to Vanish's guarantee that secret shares are widely dispersed, safe from collusion, and have high turnover. Anything less than a world-wide DHT is unlikely to be secure for Vanish because smaller communities are vulnerable to collusion attacks and social engineering .

Counter measures There are two potential directions for mitigating the Unvanish attack. One is that one could use an even "more secure" setting on the current system such as requiring 99 of 100 shares. This approach is problematic for two reasons. First, requiring such a high recovery rate will cause the system to be unusable for legitimate users as secret shares are lost via churn. Second, an attacker could react by scraping the DHT more completely, either by buying more compute resources or by improving the efficiently of the (inefficient) Vuze client code.

Another possible counter measure is to migrate the Vanish system to another DHT system that has more security, such as OpenDHT. While it is possible that this might make such an attack more difficult, we conjecture that one could adapt our attack to this setting as well. In general, we believe that any modification of the overall Vanish approach should be received with increased skepticism.

Conclusion The Vanish authors claim that Vanish cannot make security worse. Vanish provides an additional layer, which if compromised, is no worse than what the user had in place originally. This argument assumes that user's behavior will be not be affected by the perceived benefits that Vanish delivers, which seems unlikely. Why bother to prune my own data if Vanish is doing it for me? Our conclusion is that the Vanish security guarantee is useful in principle, but is not achieved by the Vanish system and cannot be achieved by a similar system.

5 An Independent Break

In personal communication with Ed Felten [6] we learned that another group independently broke the Vanish system by launching a Sybil attack.[2]

Acknowledgements

We thank Adam Klivans for a useful discussion at the an early stage of our project.

References

[1] *Azureus*. http://www.vuze.com.

[2] M. Bellare and S. K. Miner. A forward-secure digital signature scheme. In *CRYPTO*, pages 431–448, 1999.

[3] R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT*, pages 255–271, 2003.

[4] J. R. Douceur. The sybil attack. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, 2002.

[5] A. EC2. Pricing of ec2.

[6] E. Felten and Colleagues. Personal communication — will fill in information, 2009.

[7] R. Geambasu, T. Kohno, A. Levy, and H. M. Levy. Vanish: Increasing data privacy with self-destructing data. In *Proc. of the 18th USENIX Security Symposium*, 2009.

[8] G. Itkis and L. Reyzin. Forward-secure signatures with optimal signing and verifying. In *CRYPTO*, pages 332–354, 2001.

[9] P. Maymounkov and D. Mazires. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems, (IPTPS)*, 2002.

[10] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[2] We plan to provide more details after further discussion.