FirePatch: Secure and Time-Critical Dissemination of Patches

Håvard Johansen
University of Tromsø
Norway

Dag Johansen
University of Tromsø
Norway

Robbert van Renesse
Cornell University
USA

Abstract

Because software security patches relay information about vulnerabilities, they can be reverse engineered into exploits. Tools for doing this already exist. As a result, there is a race between hackers and end-users to first obtain patches. In this paper we present and evaluate *FirePatch*, an intrusion-tolerant dissemination mechanism that combines encryption, replication, and sandboxing such that end-users are able to win the security patch race.

1 Introduction

Automatic software updating over the Internet is essential for a large-scale computer infrastructure in use by hundreds of millions of users. Such "patches" are particularly important if they fix security holes. Consequently, several software companies offer patches on their websites and automatically alert users of their availability.[1] Some software even supports automatic installation of patches.

While at first sight this seems useful to the security of software, dissemination of security patches has actually resulted in a significant reduction of security. This paradox stems from the fact that a security patch can be reverse engineered to reveal vulnerable code. A malicious exploitation can be initiated on nodes that have not yet installed the patch. Hence, there is a race from when a security patch is released to when all vulnerable hosts have successfully installed the patch. Making matters worse, many machines are on-line only occasionally, and can be compromised soon after they go on-line but before they have a chance to discover and install a patch.

We call this time period the *window of vulnerability* (WOV). Recent studies are compelling. For instance, [3] reports in their study of the *Windows Update* system involving 300 million users that 20% of end-users take more than 24 hours to install a patch. This is more than enough time to construct and disseminate an exploit based on a new security patch. Tools for reverse engineering patches already exist[2], and there are even reputed to be tools that construct exploits automatically. Hackers could also mount a Denial-Of-Service (DoS) attack against the patch web site (after learning of a patch) in order to increase the WOV.

This paper describes *FirePatch*, a scalable and secure overlay network for disseminating security patches. *FirePatch* employs the following three techniques:

1. A patch is disseminated in two phases. First, an encrypted version of the patch is disseminated (which cannot be reverse engineered). Some time later, the decryption key is disseminated. As the key will typically be significantly smaller than the patch, it can be disseminated much faster to a large collection of machines.

2. In order to deal with DoS attacks that attempt to increase the WOV, we have developed a distributed software mirroring service. While replication makes

[1] Users may prefer to install a packet filter "shield" [8] that prevents a security hole from being exploited, rather than having to restart an application. Our solution is useful for either patches or shields, and we shall not make a distinction in this paper.

[2] See, for instance, Symantec and their binary analyzing toolkit `http://www.bindview.com/Services/Razor/Papers/2004/comparing_binaries.cfm`

DoS more difficult, it increases the likelihood that individual servers are compromised—a highly undesirable situation for a server that disseminates security patches to clients. Therefore, our service is also made tolerant of Byzantine failures.

3. For machines that are not on-line all the time, we have developed a simple protocol for secure download and installation of patches that is run when a machine goes on-line. While this goes on, the machine is prevented from participating in other network communication.

The rest of this paper is organized as follows. In Section 2 we outline the architecture of *FirePatch* and state our assumptions. Section 3 describes our two-phase dissemination protocol which we use in our dissemination overlay described in Section 4. Section 5 outlines the protocol for disconnected clients. *FirePatch* is evaluated in Section 6 and related work is presented in Section 7. Section 8 concludes.

2 Architecture and Assumptions

We distinguish three roles: *patchers*, *clients*, and *mirrors*. Patchers are typically software providers that issue patches. For simplicity, we will assume a single patcher in this paper, although any number of patchers is supported. Clients are machines that run software distributed by the patcher. Mirrors are servers that store patches for clients to download, and notify clients when a new patch is available.

We assume that the patcher is correct and is trusted by all correct clients. In particular, using public key cryptography clients can ascertain the authenticity of patches. In our system, clients are passive participants, and in particular do not participate in the dissemination system. Thus we do not have to assume that clients are correct.

In order to deal with DoS attacks against the patcher, we employ a distributed network of mirror servers. The more mirrors, the harder it is to mount a DoS attack against the network. However, the easier it is to compromise one or more mirrors. We allow a subset of mirrors to become compromised, but assume that individual compromises are independent of one another, and that the probability that a mirror is compromised is bounded by a certain P_{byz}.

The patcher publishes (and signs) the list of servers that it considers mirrors for its patches. This list contains a version number so the patcher can securely update this list when necessary.

We assume that all communication goes over the Internet, the shortcomings of which are well-known. In order to deal with spoofing attacks, all data from the patcher is cryptographically signed, and we thus assume that the cryptographic building blocks are correct and the private key is securely kept by the patcher.

3 Two-Phase Dissemination

We have devised a dissemination protocol that, when layered on top of a secure broadcast channel, makes the WOV independent of message size. The net result of such an invariant is that the WOV can be kept fixed and small despite the fact that voluminous data has to be transferred over the wire.

At first, this might seem as an impossible invariant. It is not, and the overall and general applicable idea is intuitively simple; we basically pre-send the varying size patch without opening the WOV. This is done by disseminating patches (or any data) in two phases. In round one, we distribute an encrypted patch, and in the second phase, we disseminate the fixed size decryption key.

The beauty of this scheme is that the WOV only contains phase two. The time between the two phases is a policy decision. One extreme is to do the second phase immediately when the first phase completes. However, this is not a viable approach as disconnected clients will delay the completion. More alarmingly, Byzantine clients will be able to stop the dissemination by claiming not to have received any messages. A better scheme is to start phase two some time after phase one is started. For instance, in the Windows Update system, a 24 hour time period between the phases would likely update at least 80% of the clients [3].

More formally, our general applicable protocol is specified as follows. Let m be a message that a source s wants to disseminate to a set of clients. In the first phase, s generates a symmetrical encryption key k and a unique identifier *UID*, and broadcasts a $\langle$ENVELOPE, *UID*, k(m)$\rangle$

message, signed by s. Upon receipt and verification of the signature, clients store this message locally. In the second phase, s broadcasts $\langle$KEY, *UID*, k$\rangle$ to all clients. Upon receipt, clients can decrypt the ENVELOPE message.

In our system the UID contains a version number so clients can distinguish newer from older version of patches.

4 Secure Dissemination Overlay

As mentioned before, *FirePatch* employs a network of mirrors to fight DoS attacks. Thus, the patcher does not broadcast patches and keys directly to the clients, but instead to the collection of mirrors. The mirrors forward this information to all clients that are currently connected to the Internet, and provides it on demand to clients that connect to the Internet at a later time.

In order to disseminate data reliably among the correct mirrors, we employ the Fireflies group membership protocol [7] combined with ChainSaw, a request-response style of gossip [6]. As mirrors can be Byzantine, the patcher and clients have to connect to a sufficient number of mirrors in order to make the probability that they are connected to at least one correct mirror node sufficiently high. With the probability of a mirror node being Byzantine is P_{byz}, if a client connects to k nodes, then the probability that all nodes are Byzantine is P_{byz}^k. If this probability is to be less than ϵ, then $k > \log \epsilon / \log P_{byz}$. (If M is the number of mirror nodes, and C the number of clients, then each mirror node will have about $C * k/M$ clients.)

Thus the patcher first disseminates the encrypted (and signed) patch to k mirrors, guaranteeing that at least one correct mirror receives the patch with a probability higher than $1 - \epsilon$. Using Fireflies and ChainSaw, all correct mirrors quickly obtain and store the encrypted patch, and each mirror notifies each of its clients. As a client is registered with at least k mirrors, it obtains such a notification with high probability. It then selects one of the mirror servers and attempts to download the patch. Should this fail, it tries another mirror server and repeats this until successful.

After some predetermined period of time, the patcher sends the signed key to k mirrors. Each correct mirror, upon receipt and verification of the signature, forwards the key to each of its registered clients and stores the key for later retrieval. A client may get multiple correctly signed copies of the key, but as the key is small this is not a problem. Using one of the copies, the client decrypts the patch (or shield), and applies it.

5 Disconnected Nodes

A tricky problem is that not all clients may be up and connected to the Internet at the time that the patch is being disseminated. When at some later time such a client connects to the Internet, it is vulnerable as hackers have now had ample time to create an exploit and may be lurking on such clients. We thus need a protocol for connecting clients to get the patches it is missing without being compromised.

Our approach is as follows. When running, clients store the list of all mirrors (disseminated by the patcher just like patches and keys) on disk. When a client connects, a local firewall is initially configured to block all network traffic except to and from $2k-1$ mirrors selected at random from the stored list. Also, the firewall only admits a limited number of message formats, as used below.

First, the client sends a $\langle$RECOVER, $v\rangle$ message to each of the mirrors, where v is the version of the latest installed patch at the client. Each of the $2k - 1$ mirrors responds with notifications of the missing patches as in the protocol described above for connected clients, and the client proceeds to download the necessary patches and keys while all other messages are ignored and dropped.

The recovering client awaits exactly k notifications before it resumes normal operation. Waiting for more is dangerous as the malicious mirror nodes may not respond and thus prevent the node from recovering. Waiting for $k - 1$ or fewer increases means that probability that all notification are from malicious mirrors becomes larger than ϵ.

6 Evaluation

Our prototype implementation is written in Python and was evaluated on a local cluster of 39 3.2 GHz Intel Prescott 64 machines with 2 GB of RAM. The machines were connected by a 1 Gbit Ethernet network. We ran 5 mirrors on each machine for a total 195 mirrors. 10% of the mirrors (chosen randomly) were configured to mount

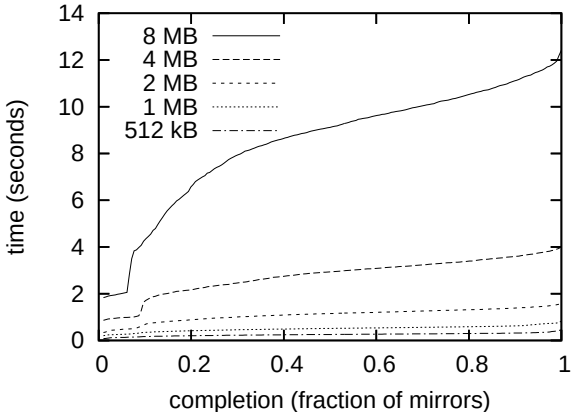

Figure 1: Completion times for varying envelope sizes.

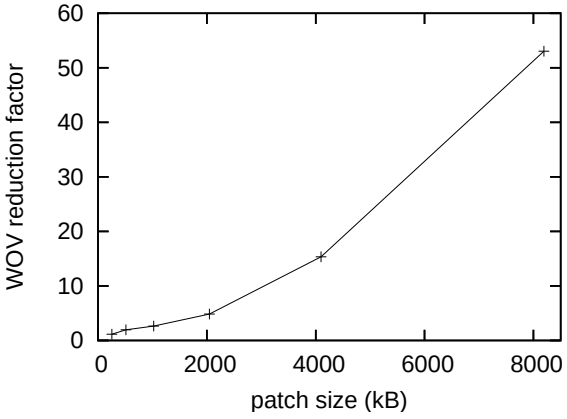

Figure 2: WOV reduction for varying patch sizes.

a DoS attack by not forwarding KEY and ENVELOPE messages.

Our experiment consisted of injecting a new patch into the dissemination overlay every 320 seconds with the corresponding encryption key released after a 300 seconds delay. We varied the size of the patches between 256 kB and 8 MB and injected 10 patches of each size. The size of decryption keys was set to 20 bytes. Mirrors logged the time when they received envelopes and keys.

Figure 1 shows the time it took for an increasing fraction of the mirrors to receive envelopes of varying sizes. For instance, a 8 MB envelope was received by 20% of the mirrors after 6 seconds and received by all mirrors after 12 seconds. The aggregate data throughput is in this case approximately 1 Gbit/s. While these numbers are pragmatic for our particular setup, they indicate that *FirePatch* will be able to saturate available bandwidth in real-world deployment.

Without our two-phase dissemination protocol the WOV depends much on the size of the patch, as can be seen from Figure 1. The time it took to disseminate a key is independent of the size of patches, and took approximately 0.23 seconds in each case. For example, for a 8 MB unencrypted patch, the two-phase protocol reduces the WOV by a factor of 53. Figure 2 shows the reduction factor as a function of the patch size.

7 Related Work

With approximately 300 million clients, Microsoft Windows Update is currently the world's largest software update service [3]. The service consists of a (presumably large) pool of servers that clients periodically pull for updates. Other commercial patch management products like ScriptLogic's Patch Authority Plus[3] and PatchLink Update[4] enable centralized management of patch deployment. It is, however, unclear how any of these systems protect themselves from intrusion and if they provision for the ability of hackers to reverse-engineer patches into exploits.

Open-source communities, like the Debian GNU/Linux Project[5], organize their software update services similarly to Windows Update, as a pool of servers that clients periodically pull for updates. Clients can freely choose which server to pull. The servers are organized in a hierarchy with children periodically querying their parent for updates. As these communities rely on donated 3rd party hosting capacity, malicious entities can easily intrude into the server pool.

SplitStream, Bullet, and Chainsaw [1, 6, 5] are efficient peer-to-peer content distribution systems that achieve high throughput by spreading the forwarding load to all

[3] http://www.scriptlogic.com/products/patchauthorityplus/

[4] http://www.patchlink.com/

[5] http://www.debian.org

peers. While the elimination of dissemination trees in Chainsaw makes it more robust than SplitStream and Bullet to certain failures, these systems do not tolerate Byzantine failures. SecureStream [4] provides Byzantine tolerant dissemination by layering a Chainsaw style gossip mesh on top of the Fireflies membership protocol [7] similarly to *FirePatch*. However, SecureStream targets live streaming of multimedia.

Vigilante [2] is a collaborative worm detection system that automatically generates self-certifying alerts (SCAs) upon worm detection. SCAs are similar to Shields [8] in that, when applied at the clients, they prevent worms from exploiting vulnerable software. It is unclear if SCAs or Shields can be reverse-engineered into exploits. Both can be disseminated by *FirePatch*.

8 Conclusion

We have investigated an approach to securely distribute software security updates in partially connected Internet environment, combining encryption, replication, and sandboxing upon reconnection of disconnected computers. Our findings are intuitive, but are highly effective.

Currently, we are experimenting with *FirePatch* on PlanetLab to achieve more realistic experience. Also, we are porting our push-based Debian Patch Dissemination toolkit [9] to *FirePatch*.

Availability

The *FirePatch* code is currently available in Fireflies' CVS repository on SourceForge (`http://sourceforge.net/projects/fireflies`).

References

[1] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. SplitStream: high-bandwidth multicast in cooperative environments. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 298–313, New York, NY, USA, 2003. ACM Press.

[2] Manuel Costa, Jon Crowcroft, Miguel Castro, Antony Rowstron, Lidong Zhou, Lintao Zhang, and Paul Barham. Vigilante: end-to-end containment of internet worms. In *Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 133–147, New York, NY, USA, 2005. ACM Press.

[3] Christos Gkantsidis, Thomas Karagiannis, Pablo Rodriguez, and Milan Vojnović. Planet scale software updates. Technical Report MST-TR-2006-85, Microsoft Research, June 2006.

[4] Maya Haridasan and Robbert van Renesse. Defense against intrusion in a live streaming multicast system. In *Proceedings of the 6th IEEE International Conference on Peer-to-Peer Computing*, Cambridge, UK, September 2006.

[5] Dejan Kostić, Adolfo Rodriguez, Jeannie Albrecht, and Amin Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *Proceedings of the 19th ACM Symposium on Operating System Principles*, October 2003.

[6] Vinay S. Pai, Kapil Kumar, Karthik Tamilmani, Vinay Sambamurthy, and Alexander E. Mohr. Chainsaw: Eliminating trees from overlay multicast. In *Proceedings of the 4th International Workshop on Peer-to-Peer Systems*, volume 3640 of *Lecture Notes in Computer Science*, pages 127–140, Ithaca, NY, USA, 2005. Springer.

[7] Håvard Johansen, André Allavena, and Robbert van Renesse. Fireflies: Scalable support for intrusion-tolerant network overlays. In *Proceedings of Eurosys 2006*. ACM European Chapter, April 2006.

[8] Helen J. Wang, Chuanxiong Guo, Daniel R. Simon, and Alf Zugenmaier. Shield: vulnerability-driven network filters for preventing known vulnerability exploits. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 193–204, New York, NY, USA, 2004. ACM Press.

[9] Ole-Petter Wikene. Distributed, intrusion tolerant, push-based patch dissemination. Master's thesis, University of Tromsø, Norway, June 2006.