Application-Level Reconnaissance: Timing Channel Attacks Against Antivirus Software

Mohammed I. Al-Saleh
University of New Mexico
Department of Computer Science
Mail stop: MSC01 1130
1 University of New Mexico
Albuquerque, NM 87131
alsaleh@cs.unm.edu

Jedidiah R. Crandall
University of New Mexico
Department of Computer Science
Mail stop: MSC01 1130
1 University of New Mexico
Albuquerque, NM 87131
crandall@cs.unm.edu

ABSTRACT

Remote attackers use network reconnaissance techniques, such as port scanning, to gain information about a victim machine and then use this information to launch an attack. Current network reconnaissance techniques, that are typically below the application layer, are limited in the sense that they can only give basic information, such as what services a victim is running. Furthermore, modern remote exploits typically come from a server and attack a client that has connected to it, rather than the attacker connecting directly to the victim. In this paper, we raise this question and answer it: Can the attacker go beyond the traditional techniques of network reconnaissance and gain high-level, detailed information?

We investigate remote timing channel attacks against ClamAV antivirus and show that it is possible, with high accuracy, for the remote attacker to check how up-to-date the victim's antivirus signature database is. Because the strings the attacker uses to do this are benign (*i.e.*, they do not trigger the antivirus) and the attack can be accomplished through many different APIs, the attacker has a large amount of flexibility in hiding the attack.

1. INTRODUCTION

Network reconnaissance is a vital step for the remote attacker before launching an attack. Attacking every reachable host is not desirable to the attacker because only vulnerable hosts can be successfully penetrated. Port scanning is a well-known technique that provides the attacker with very useful information about possible victims. The attacker wants to know if the victim is running certain services, and determines this by sending packets to certain ports the victim might be listening on. The communication between the attacker and victims could reveal the victims' specific services and operating system, and even version information. Port scanning, while helpful to the attacker, is limited by the kind of information it can attain and by the ability to reach victims that have contacted the attacker from behind Network Address Translation (NAT). Stateful firewalls and intrusion detection systems, if well deployed and configured, are considered strong defense lines against port scanning. Also, maintaining stealth is important. In this paper, we explore techniques to remotely gain detailed high-level information about a victim host that has connected to the attacker's web server. Our goal is to better understand how the future network reconnaissance techniques that network defenders must anticipate will work.

1.1 Application-level reconnaissance

We are particularly interested in the ability of the remote attacker to learn about victims beyond traditional techniques. The highest level of information the attacker can obtain is information related to the victims' running applications, particularly security-related applications. Vulnerable applications, firewalls, and antivirus software are of interest to the attacker. We take the antivirus application in our study as a running example and show how the attacker can stealthily gain information about the victims' antivirus. In this example, the attacker creates a timing channel to infer how up-to-date the antivirus is. This could be useful in that the attacker could decide to send older versions of malicious code and thereby limit the exposure of their newest version by only using it when necessary. In traditional network scanning, the remote attacker scans victims by the means of being a client who tries to connect to servers and discover vulnerabilities. In our model, the attacker is modeled as a server who waits for clients' connections and then scans them at the application layer. This model is particularly suited to drive-by-downloads.

1.2 Threat model

While modern antivirus software employs advanced techniques such as filtering, algorithmic scanning, and emulation, at its heart antivirus is still based on pattern matching. Note that the filters that trigger algorithmic scanning and emulation are still pattern-based, and advanced techniques also amortize their performance and therefore offer opportunities for timing channels.

An antivirus scanning engine scans data against its virus signatures. The antivirus does not compare data to every single signature sequentially, but rather it stores the signatures in data structures that allow for fast scanning that is optimized for the common case (typical strings of bytes) and amortizes the performance overhead by having slow code paths that are only taken when a byte string is close to a signature in the database in some way. Depending on how the antivirus stores the signatures in the data structures, scanning one piece of data can take a longer time than another based on the scanning path the antivirus takes to determine if the data is malicious or benign. Suppose that the attacker knows how the antivirus scanning works, then they can create special crafted data

that makes the antivirus take a longer time if a certain signature is in the database, but less time otherwise. In our threat model, the attacker wants to know if a client's antivirus database is updated with a certain signature or not. Although the attacker is modeled as a server in our threat model, she can be modeled as a malicious insider as well.

The fundamental principle the antivirus software utilizes is making the common case fast. However, this introduces the possibility of timing channel attacks.

1.3 Why antivirus?

Having antivirus software is considered essential for typical computers today. According to a study [4], 81 percent of users use antivirus on their computers. Antivirus signature databases vary widely in terms of how up-to-date they are, due to both users who have not updated recently and scaled releases of updates. An attacker need not use a more recent malicious code, and thereby increase the exposure of the more recent code, if a user's antivirus signatures are not up-to-date and an older malicious code will suffice.

We chose ClamAV antivirus in our study because it is an open source antivirus.

1.4 Why timing channels?

The benefit of a timing channel attack is flexibility and stealth. Even though the attacker might be able to directly check how up-to-date the antivirus' database files are through ActiveX controls or other APIs that allow direct checking of directories, files, and processes, this suspicious behavior will have a distinct behavioral signature that is difficult for the attacker to obfuscate (*e.g.*, opening the antivirus' signature database). Also, the database files could be hidden or not allowed to be reached by the attacker in the first place. So, indirectly inferring how up to date the database is is preferable from the attacker's point of view.

1.5 Paper structure

This paper is organized as follows. First, we give a background of how ClamAV signature scanning works in Section 2. This is followed by Section 3 that explains our evaluation methodology, and then our results in Section 4. A discussion and future work are in Section 5. Then related works and the conclusion follow.

2. BACKGROUND

2.1 On-access vs. on-demand scanning

On-access scanning is triggered upon file system operations, such as ***open***, ***create***, or ***close*** system calls. To be scanned with the on-access scanner, a virus should be read from or written to the disk. On-access scanners run as daemons and hook into the file system APIs or are implemented as device drivers that are attached to the file system [21]. On-demand scanning starts only if the user asks the scanner to scan some files.

2.2 ClamAV antivirus

ClamAV [2] is a well-known, open source antivirus program. ClamAV consists of a main library and a set of command line programs that make use of the APIs provided by the library. On-access scanning in Windows is possible via Clam Sentinel (see below).

2.2.1 File type filtering

The ClamAV scanning engine has 10 different roots which correspond to 10 different file types. These are GENERIC, PE, OLE2, HTML, MAIL, GRAPHICS, ELF, ASCII, NOT USED, and MACH-O. ClamAV signatures are loaded into the data structures of those roots depending on what kind of files a virus can infect. For example, if virus **X** infects PE files and a signature **X'** is generated for **X**, then **X'** will be loaded into the data structures of the PE root. When ClamAV scans a file, it checks the file type first to determine which root's signatures will be used to scan that file against. File type filtering speeds up the scanning process by optimizing scans for file types where the entire file need not be scanned.

2.2.2 Filtering step

To make scanning even faster, ClamAV implements an additional filtering step prior to scanning. Every type root has its own filter. The filter can determine if a file is benign before scanning it. The most important feature of these filters is that they do not have false negatives but do have false positive. In other words, the filter will not let a file containing a virus pass without being scanned, but if it can prove that the file is benign then no further scanning is needed for that file. However, some other benign files might cause a hit in the filter and thus will need to be scanned further. ClamAV implements a bit-level state-machine to match characters in the filter. The state machine has 8 states where each state is represented by 1 bit. The state machine might have multiple active states at the same time and thus multiple transitions might be taking place in parallel. Because ClamAV checks input against the filter, any character can be good to start checking from. Thus, ClamAV activates state 1 at each transition. An active state 1 is represented by 11111110. The filter is an array, called B, of length 65536, where each element is 8 bits long. Figure 1 illustrates this. ClamAV chooses 8 characters carefully from each signature to add it to the filter. Then, it iterates through the 8 characters and reads q-grams of length 2 at each position. For example, if the 8-byte string of a signature is 0x001122334455667788, then it changes the filter as in the following steps[1]:

1. For position 1 of the string, execute B[0x0011] = B[0x0011] & 11111110. This says that 0x0011 is satisfactory to start with.
2. For position 2: B[0x1122] = B[0x1122] & 11111101.
3. It continues in the same way for the following positions until the 7th position.
4. To mark the end of a string ClamAV has another array called End. For the previous example, when position 7 is reached, the End array is changed to be End[0x7788] = End[0x7788] & 10111111.

After ClamAV determines which type root an input belongs to, it checks the root's filter against the input. Searching the filter starts by setting the bit-level state machine to 1111111 (no active states). Then, ClamAV iterates through all input characters until it finds a match or reports a negative result. At each character position, ClamAV reads 16 bits as q0 (q-gram equals 2) and performs this statement:

[1] <<, |, and & represent shift left, or, and and bitwise operations, respectively

state = (state << 1) | B[q0], where state << 1 activates state 1 each time, and then it checks if it finds a match by this statement: match_state_end = state | End[q0], and reports a match if match_state_end != 0xff. In other words, it reports a match when reaching a state at which the input can end at while being in an active state. Because this filter is created by applying this procedure to all of the signatures, a string will not pass the filtering step unless it cannot possibly match any signature.

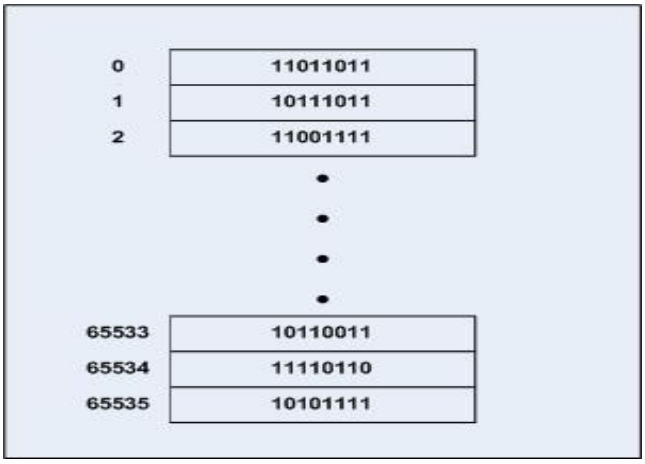


Figure 1: ClamAV filter. The filter content is based on the signatures. An active position is represented by 0. The right most bit is the first position.

2.2.3 *Aho-Corasick algorithm*

ClamAV uses an extended version of Aho-Corasick algorithm [5]. ClamAV usually uses this algorithm for signatures that have wildcards. In this case, the signature is divided into patterns that need to be matched in order to report a match. This algorithm is used to match an input against many patterns at the same time. ClamAV uses a tree-like data structure, called a trie, to store patterns. Each node in this data structure has 256 transition pointers. Each transition represents an ASCII character. What distinguishes this data structure is that each node has the same prefix as its predecessor nodes up to the root node, so that all patterns with the same prefix will take the same path. Because each node takes a considerable amount of memory and also because the trie could be very large if not restricted, ClamAV has put a maximum limit for the depth to be 3. After reaching the maximum depth, all patterns will be added to a linked list attached to leaf nodes. Suppose that there is a signature that starts with the sequence 0x010002. Figure 2 shows the transitions in the ClamAV trie structure that this sequence would take. Once the leaf node is reached, the pattern will be added to the linked list of patterns which have the same prefix. Also, fail transitions are established, so that instead of returning back to the root at every mismatch, a fail transition is made to the proper node. For example, if the next input at node 2 is 0x01, then the fail transition is set to go back to node 2 instead of re-matching starting again from node 1.

2.2.4 *Boyer-Moore Algorithm*

An extended version of Boyer-Moore algorithm [10] is used in ClamAV. This algorithm is usually used for signatures which do not have wildcards. The original Boyer-Moore algorithm scans patterns against input from right to left. Two tables are used to determine how many characters the pattern needs to be shifted by.

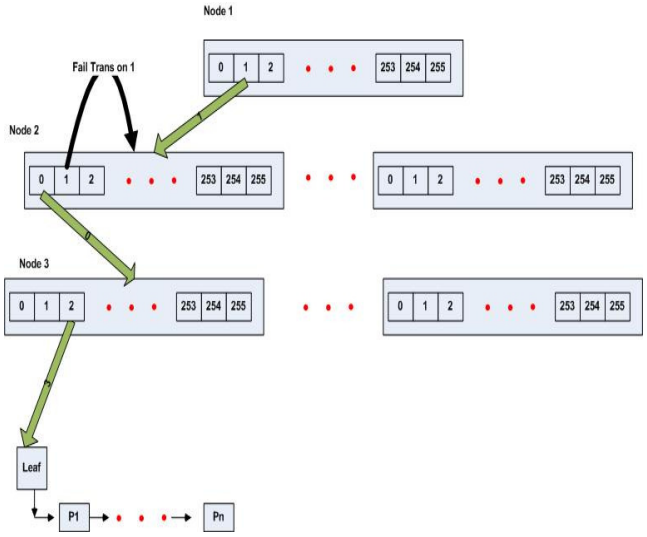


Figure 2: ClamAV Aho-Corasick trie structure with arbitrary success transitions and one fail transition. Each transition represents an ASCII character. The maximum depth is 3. Patterns are added to linked lists after bypassing the maximum depth.

The two tables are built based on two roles, the bad character shift role and the good suffix shift role. For this algorithm, ClamAV uses an array of linked lists to distribute the signatures among. To determine a location for a signature in the array, ClamAV hashes 3 characters of the signature using a hashing function. ClamAV tries to evenly distribute signatures in the array by trying the next 3-character sequence of the signature if the first 3 collides, and so on.

2.3 ClamWin and Clam Sentinel

ClamWin is a free antivirus for Microsoft Windows and used by more than 600,000 users worldwide on a daily basis [3]. It is based on the ClamAV scanning engine. ClamWin only supports on-demand scanning. Clam Sentinel is a free program that works with ClamWin to support on-access scanning.

3. EXPERIMENTAL METHODOLOGY

Our experimental methodology was designed to answer the following two questions:

Question #1: Is there an exploitable timing channel based on how new signatures are added to ClamAV database? The basic idea behind the timing channel we demonstrate is to make the scanning engine hit in the place in which a signature is added over and over again to add a measurable delay to the scan. If the signature is there, then more work will take place and this means more scanning time. Three things are involved to make this happen. **First**, for any input to be scanned against database signatures, it needs to pass the filtering step. We extracted the exact 8 characters from each signature that are added to the filter, see Section 2.2.2 for more details about how the filtering works. When scanning files, ClamAV divides files into buffers of length 128KB. We need to make sure to plant the extracted characters in a buffer size basis rather than a file size basis. **Second**, for signatures that are added to the Boyer-Moore linked list, we extracted the characters from each Boyer-Moore signature that is used in the hashing function to add the signature to the linked lists, see Section 2.2.4 for more

details about how the Boyer-Moore algorithm works. **Third**, for signatures that are added to the Aho-Corasick trie structure, we extracted the characters from each Aho-Corasick signature that will cause the scanner to go all the way down to the leaf nodes. See Section 2.2.3 for more details about how the Aho-Corasick algorithm works.

To demonstrate a possible timing channel, we collected the names of all of the viruses added since the first available ClamAV release, which was 17 April 2004. ClamAV maintains a virus database mailing list through which it reports virus addition or update. We downloaded and parsed all the HTML files since that date and put the results in a name-date list. Then, we unpacked the ClamAV database and removed from it all of the signatures of viruses that have their names on our list. This step makes the database as if it is the database from 17 April 2004. We made two kinds of experiments: the day-basis experiment and the signature-basis experiment. For the day-basis experiment, we wrote a script that creates files per date/day. For example, if in date **D** **S** signatures are added to the database, then the script will create the Directory **D**. Inside **D** the script creates **S** directories each one corresponds to a virus signature. In each directory of **S** directories, the script creates 10 files each of which is 1MB. See Figure 3. The number and size of the files are below the default ClamAV limits to scan files. The 10 files' contents are identical. To create a file, we concatenate the extracted Boyer-Moore or Aho-Corasick characters from the corresponding signature until it reaches 1MB size. Meanwhile, we plant the filter characters of the signature every time we reach the buffer size. The initial content of the file depends on the file type a signature is taken from, see Section 2.2.1. The test begins by scanning the oldest date directory we have, then we add the signatures of that date to the database and scan it again. Then we scan the next date directory before and after adding that date's signatures, and so on. For the signature-basis experiment, we tested to see the effect of adding single signatures rather than the signatures for the whole day. For this experiment, ClamAV is asked to scan the 10 files, which consist of a signature's extracted characters, before and after adding that signature to the database.

Question #2: If the first question is confirmed, is it possible to exploit the timing channel in a real attack? Figure 4 illustrates a real-world scenario. A victim, who has ClamWin antivirus and Clam Sentinel installed, connects, through Internet Explorer, to a web server which is controlled by the attacker. Once connected, the victim is asked to download an ActiveX component that looks necessary to accomplish a certain task. Once downloaded, the ActiveX is started by JavaScript code. Then the ActiveX component creates a file that will be scanned by the antivirus and measures the CPU usage for a certain amount of time to determine the busy period the CPU experienced. We used the PDH (Performance Data Helper) library to query the processor time performance counter. This experiment starts when the CPU is almost idle. To determine the busy period of the CPU, we need a way to make sure that the CPU is busy with the antivirus rather than any other process which might be woken up or started at some time and then compute for a small amount of time. So, the attacker can repeat the process to separate signal from noise coming from the CPU running other processes. What distinguishes the antivirus scanning process is that it keeps the CPU busy until it is done. This feature makes differentiating the start and the end time of the busy period easier. We determine the busy period in two stages: in the first stage, we start collecting all CPU usage samples (1 data point every 15 milliseconds) for 35 seconds, which is more than enough for the antivirus to scan a file, starting right after we close the file (the on-access scanner starts scanning right after closing the file). In the second stage, we take the averages of every contiguous 10-data-point sequence of the collected data from the first stage. We take the higher-than-normal averages (a threshold is set empirically) as start and end points and compute the elapsed scanning time based on this difference. After getting the total time the antivirus spent scanning the file, the ActiveX component triggers an event that will be received by a JavaScript function which, in turn, will send the results to the remote attacker.

A simpler model could be an insider threat which does not need a connection to a remote server.

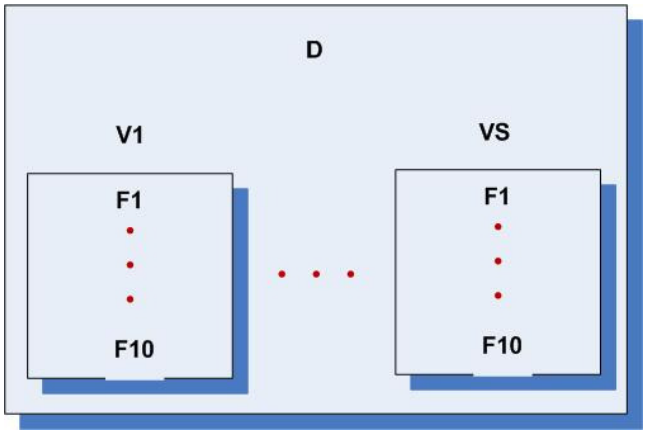


Figure 3: Test file hierarchy per date D. The higher level directory D contains the files created for all signatures which were released on that day. Each V directory contains files created for only one signature.

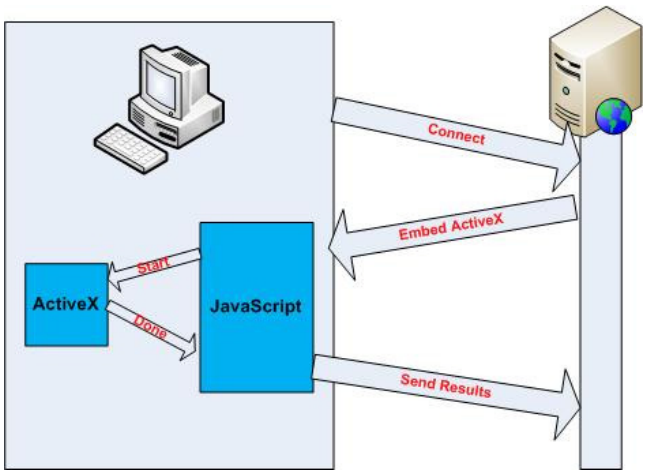


Figure 4: A scenario for a real-world attack. The client uses Internet Explorer to connect to the server. Then, the client is asked to download an ActiveX component which a JavaScript script can control. The component creates a file and returns the CPU busy period, which will be considered as the scanning time, to the JavaScript as an event. The JavaScript sends the result back to the server.

4. RESULTS

In this section, we present results to answer the two questions we asked in Section 3. The results show that an attacker can exploit a timing channel from newly added signatures and that this attack can be implemented in a real-world scenario. We ran the first two experiments in Linux 2.6, on an Intel Core 2 Quad CPU at 2.66 GHz, and 8 GB RAM. Also, we ran the ActiveX experiments in Windows 7 OS, on an Intel Dual Core Atom CPU at 1.66 GHz, with 4 GB RAM.

4.1 Day-by-day experiment

In this experiment, we scanned a number of files that were created for each day, before and after adding the signatures of the day. See Section 3 for more details about the setup. The **clamscan** command line program was used to initiate the scan. The averages were taken over 10 runs for each day. We take the differences between the scanning time averages after and before adding signatures (*i.e.*, the time to scan files after adding the signatures minus the time to scan files before adding the signatures).

Figure 5 shows a histogram of the differences represented by the ranges. For an x-axis value v, the corresponding y-axis value w is the number of occurrences that are in this range: $(v - 0.025: v]$. As expected, there are only 10 instances where the scanning times before and after adding signatures are almost the same. For all other instances, adding new signatures creates timing differences.

4.2 Single signatures experiment

In this experiment, we scan a number of files that are created to exploit only one signature each time. The **clamscan** command line program was used to initiate the scan. The averages were taken over 4 runs for each experiment. We take the differences between the scanning time averages after and before adding a signature. Figure 6 shows that a timing channel for one signature can be determined with high probability. For an x-axis value v, the corresponding y-axis value is the number of occurrences that are in this range: (v - 0.025: v].

4.3 ActiveX experiment for GENERIC type files

In this experiment, the ActiveX component creates 5 MB files and measures the time that the antivirus (ClamWin and Clam Centinal) spends on the CPU while scanning the files. Each run represents the creation and measurement time for only one file. This file is created by concatenating a randomly generated benign sequence of characters (*i.e.*, sequence of characters which has no effect in how a signature is inserted into Aho-Corasick tries or Boyer-Moore linked lists) or by concatenating a randomly chosen extracted sequence of characters (*i.e.*, sequence of characters which affects the shape of the Aho-Corasick tries or Boyer-Moore linked lists). The files considered are of type GENERIC. Each run is repeated 5 times to ensure that the results are consistent. Figure 7 presents the averages over 5 runs for each file for both benign and extracted cases. Figure 8 zooms out on the area where both get closer. The results show that the scanning time for the extracted characters are always higher than that of the benign characters except for one data point (data point 2 in the case of extracted and 9 in case of benign), where both are equal.

In Figure 9, we present the worst case scenario where we compare the minimum runs in the case of the extracted characters to the maximum runs in the case of benign files. The results still show that the scanning time for the extracted characters are always higher than that of the benign ones except for two data points (data points 2 and 3 in the case of extracted and 9 in the case of benign). Figure 10 zooms out on the area where both get closer.

4.4 ActiveX experiment for HTML type files

In this experiment, instead of starting by filling characters into an empty file, we append characters to a basic HTML file. This would make the numbers different from the above experiment because the scanning engine will be directed to scan the files against signatures in the HTML (rather than GENERIC) root, see Section 2.2.1 for more information. Figure 11 shows the averages over 5 runs for each created file while Figure 12 represents the worst case scenario where we compare the minimum runs in the case of the extracted characters to the maximum runs in the case of the benign files.

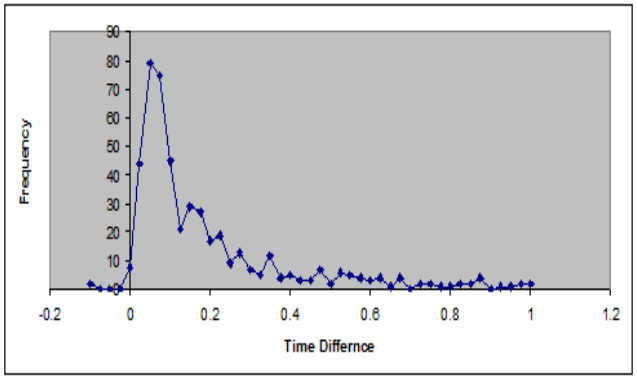


Figure 5: Scanning time differences before and after adding signatures of a day.

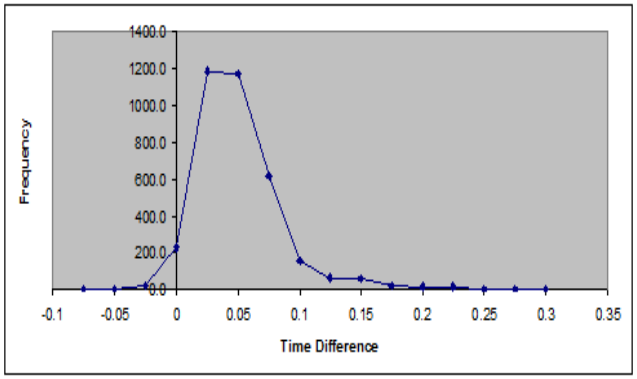
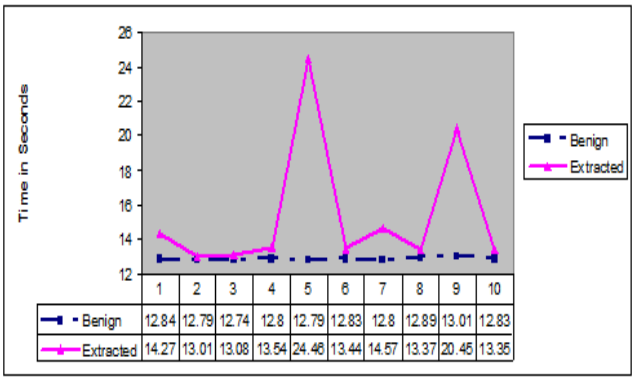


Figure 6: Scanning time differences before and after adding a single signature.

5. DISCUSSION AND FUTURE WORK

We argue that stealthily scanning clients is powerful because, besides being stealthy, it allows the attacker to gain a higher level of information. The notion of scanning clients, rather than servers, is a more fitting threat model for today's exploits that are based on "drive-by downloads." The user makes the initial connection and thus gives the attacker an opportunity to scan their machine. Web browsers are well known to be one of the main avenues for modern attack.

The running example of checking, through a timing channel, how up-to-date ClamAV is shows that application-level reconnaissance



	1	2	3	4	5	6	7	8	9	10
Benign	12.84	12.79	12.74	12.8	12.79	12.83	12.8	12.89	13.01	12.83
Extracted	14.27	13.01	13.08	13.54	24.46	13.44	14.57	13.37	20.46	13.35

Figure 7: Scanning time of creating GENERIC type files out of benign and extracted characters. Each data point represents an average over 5 runs

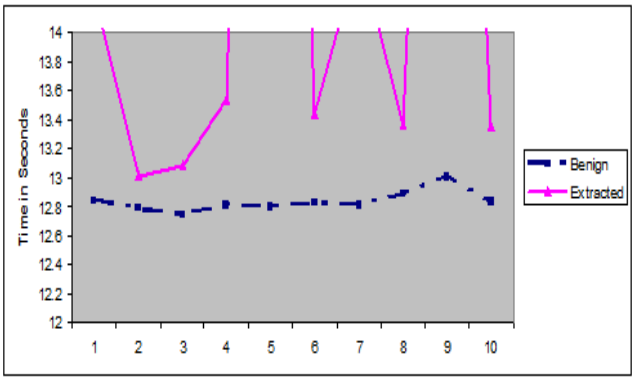
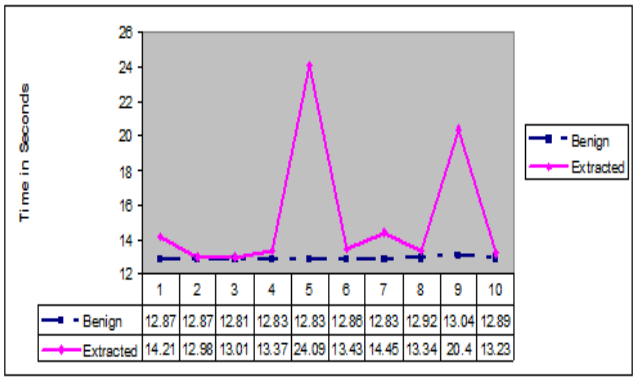


Figure 8: The same experiment as in Figure 7 with clear border between benign and extracted.



	1	2	3	4	5	6	7	8	9	10
Benign	12.87	12.87	12.81	12.83	12.83	12.86	12.83	12.92	13.04	12.89
Extracted	14.21	12.98	13.01	13.37	24.09	13.43	14.45	13.34	20.4	13.23

Figure 9: The same experiment as in Figure 7, but we show the worst case scenario.

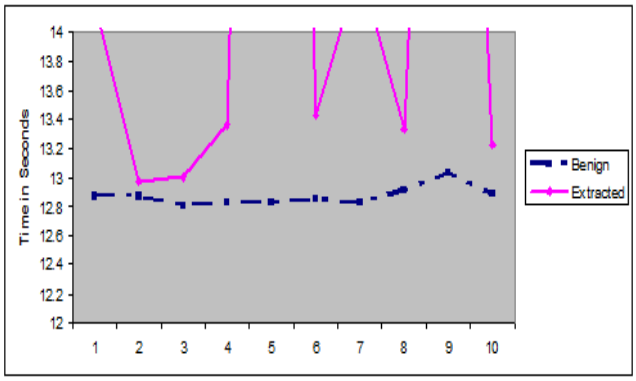


Figure 10: The same experiment as in Figure 9, but we show the worst case scenario with clear border between benign and extracted.

attacks are practical and can reveal high-level information about a user's system. The same general approach could be used to detect which antivirus program is installed (if any), the presence of other security software such as local intrusion detection systems or personal firewalls, if other malicious code is installed that hooks into some system behavior, mouse activity, the patch-level of the operating system, and more. Because all of these can be determined through timing channels, the attacker has a high degree of flexibility in the APIs used for scanning and need not have control of the system before doing reconnaissance.

Although our prototype attack, which was built for testing purposes to determine the time scales involved, is based on ActiveX the only three capabilities needed for a real application-level reconnaissance attack are to create or modify a file, measure the CPU usage, and keep track of time. Creating or modifying a file is possible even without APIs, simply by causing a file to be cached or a string of data to be logged. Furthermore, modern antivirus programs scan much more than just files and hook into a browser in many more places, so the surface available to scan them is larger than that of ClamAV. Measuring CPU usage can often be accomplished with several different APIs, or the attacker can simply measure the progress of one or more threads, which can be low priority need not necessarily consume all cores of the CPU if their timing for giving up the CPU is carefully orchestrated. Finally, keeping track of time is possible with a large variety of APIs, and need not be done on the client being scanned since the attacker's server can simply view events from the client and measure time itself. Crosby *et al.* [14] demonstrate that timing attacks can be performed over the internet with an accuracy of 15-100 microseconds, so in some cases it is not even necessary to use the clients timekeeping API.

We chose ClamAV for our tests because it is open source, the details of its core scanning algorithms are documented on the web, and the developer community for ClamAV is very helpful. We believe that more advanced antivirus programs, such as the proprietary antivirus programs that perform advanced filtering, emulation, algorithmic scanning, and heuristics, offer a lot more information that can be inferred from timing attacks than ClamAV. Table 1 shows how timing channel attacks could be possible in modern antivirus software. In general, the more sophisticated an antivirus program is, the more tradeoffs between performance and the number of patterns that can be detected must be traded off by amortizing fast codes paths *vs.* slow code paths. So, even though a closed-source, proprietary antivirus program would take a significant amount of more effort to develop timing attacks for, the opportunities for gaining detailed high-level information about, for example, how up-to-date the signature database is will be much greater.

In terms of mitigation strategies to help ameliorate the type of high-level reconnaissance attacks we present in this paper, strategies are

Technique	Description	Fast Scanning	Slow Scanning	Example
Algorithmic Scanning	Essential part of modern AV architecture. Implemented as java-like p-code (portable code) using a virtual machine	The input file does not trigger the algorithmic scanning	The file triggers the algorithmic scanning to run portable code (p-code) and its virtual machine which is hundreds of times slower than native machine code	When triggered, the algorithm to detect Zmist virus needs to execute at least 2 million p-code-based iterations [21]
Code Emulation	Powerful technique that emulates the execution through the CPU and the memory	The input file has no kind of encryption or suspicious patterns to trigger the emulation	Certain encrypted files can trigger the emulation	Fabi.9608 encrypted virus puts itself at the entry point of PE infected files. Although emulating the entry point of the infected file would expose the virus, it will significantly slow things down [21]
Heuristics	The structures of programs could trigger heuristics detection if they look suspicious	Programs' shapes look normal	Inconsistency between meta data and actual data in programs, or abnormality in the organization of program sections	Heuristic scanning is triggered when a PE program has an entry point pointing not to any of the sections but to an area after the header and before raw data (CIH-style viruses) [21]

Table 1: Modern antivirus techniques still make timing channels possible.

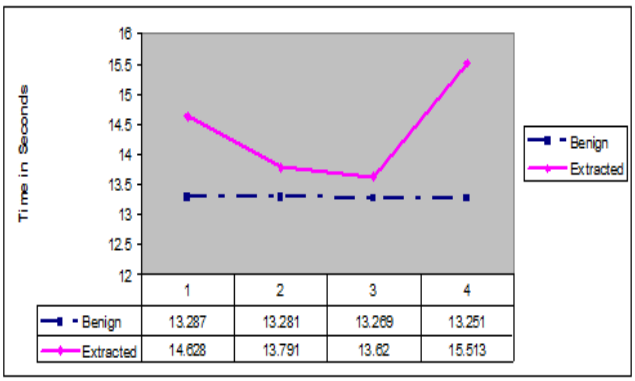


Figure 11: Scanning time of creating HTML type files out of benign and extracted characters. Each data point represents an average over 5 runs.

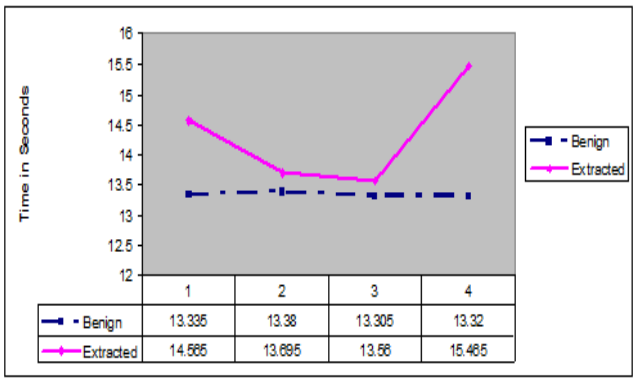


Figure 12: The same experiment as in Figure 11, but we show the worst case scenario.

needed that still allow for performance tradeoffs to be made for common cases. Predictive black-box mitigation [6] is a promising approach, and could potentially be applied by the antivirus program to maintain good performance while also minimizing the impact of timing channels attacks.

6. RELATED WORK

6.1 Network discovery

Network reconnaissance is very important first step in launching an attack. vPort scanning is a well known technique to probe networks and discover information about remote systems and the services they run. The threat model in port scanning is that the attacker, as a client, initiates the scanning victims who are servers that are reachable on the network. Nmap [18] is a well known tool to discover open, closed, and filtered ports, operating systems, services, and version information. All information that Nmap can get, however, is limited because it can only use raw packets and exposed services to discover information about networked hosts. Also, stateful and well configured firewalls and intrusion detection systems can stop many port scanning techniques. In our approach, the threat model is different. The attacker, who acts a server, waits for connections from victims, who are clients. Once connected, the attacker seeks to learn about the victim from the application layer, where a richer amount of information is available.

6.2 Timing channel attacks

Timing channel attacks are based on measuring the time it takes for a program to perform a task [7, 17, 22]. Timing channels have been exploited to reveal the secret keys in cryptographic systems [8, 19, 11], reveal SSH user passwords [20], breach users privacy [15], or detect virtual machines [16]. In this paper, we exploited a timing channel attack in ClamAV antivirus by noticing the affect of adding a signature to the database on the scanning engine algorithm.

The work closest to our own is Bortz *et al.* [9]. Using timing attacks on web applications, they were able to find out private information about a user's web activity, such as the status and result of logins and login attempts, the number of objects on a page, and so forth. Our work extends timing attacks through the browser beyond web applications and shows that it is possible to find out security-relevant information about a potential victim machine's configuration.

6.3 Antivirus research

Attacking antivirus software is possible because antivirus is just software that could have vulnerabilities [1]. Because antivirus programs match data against a signature database, evading detection is possible using obfuscation transformations [12]. New signatures for obfuscated versions of viruses are generated based on samples of the newly obfuscated versions of that virus that are found in the wild. By measuring how up-to-date a potential victim's antivirus signature database is before attacking, an attacker can use older versions of their malicious code when possible and greatly reduce the exposure of their newest malicious codes.

Christodorescu *et al.* [13] shows that it is possible to extract the signature for a specific virus that the antivirus is using to detect that virus.

7. CONCLUSION

We showed that application-level reconnaissance through timing channels has the potential to reveal detailed, high-level information about a system to an attacker. The running example we used for the experimental results given in this paper was based on checking how up-to-date the ClamAV antivirus on a given machine is. The results show that the attacker, with high accuracy, can determine if the database has been updated with certain signatures or not. Although most research concentrates on the potential of clients scanning servers, we concentrated in this paper on the possibility of scans that a server might perform on a client. Also, the scans we considered occur at the application layer and can reveal much more information than, for example, port scanning. We believe that this type of reconnaissance will become increasingly important to study in the near future.

8. ACKNOWLEDGMENTS

We would like to thank Török Edwin, a ClamAV developer, for his help in reading and understanding the ClamAV code. We would also like to thank the LEET reviewers for their very helpful comments. This work was supported in part by the U.S. National Science Foundation (CNS-0905177). Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

9. REFERENCES

[1] Attacking antivirus. `http://www.blackhat.com/presentations/bh-europe-08/Feng-Xue/Whitepaper/bh-eu-08-xue-WP.pdf`.

[2] Clam antivirus. `http://www.clamav.net`.

[3] Free antivirus for windows. `http://www.clamwin.com`.

[4] Internet security threats will affect u.s. consumers holiday shopping online. `http://www.bsacybersafety.com/news/2005-Holiday-Online-Shopping.cfm`.

[5] A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975.

[6] A. Askarov, A. C. Myers, and D. Zhang. Predictive black-box mitigation of timing channels. In *CCS '10: Proceedings of the 17th ACM conference on Computer and communications security*, New York, NY, USA, 2010. ACM.

[7] H. Bar-El. Introduction to side channel attacks.

[8] D. J. Bernstein. Cache-timing attacks on aes. Technical report, 2005.

[9] A. Bortz, D. Boneh, and P. Nandy. Exposing private information by timing web applications. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 621–628, New York, NY, USA, 2007. ACM.

[10] R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Commun. ACM*, 20(10):762–772, 1977.

[11] D. Brumley and D. Boneh. Remote timing attacks are practical. In *In Proceedings of the 12th USENIX Security Symposium*, pages 1–14, 2003.

[12] M. Christodorescu and S. Jha. Static analysis of executables to detect malicious patterns. In *SSYM'03: Proceedings of the 12th conference on USENIX Security Symposium*, pages 12–12, Berkeley, CA, USA, 2003. USENIX Association.

[13] M. Christodorescu and S. Jha. Testing malware detectors. *SIGSOFT Softw. Eng. Notes*, 29(4):34–44, 2004.

[14] S. A. Crosby, D. S. Wallach, and R. H. Riedi. Opportunities and limits of remote timing attacks. *ACM Trans. Inf. Syst. Secur.*, 12(3):1–29, 2009.

[15] E. W. Felten and M. A. Schneider. Timing attacks on web privacy. In *CCS '00: Proceedings of the 7th ACM conference on Computer and Communications Security*, pages 25–32, New York, NY, USA, 2000. ACM.

[16] J. Franklin, M. Luk, J. M. McCune, A. Seshadri, A. Perrig, and L. van Doorn. Remote detection of virtual machine monitors with fuzzy benchmarking. *SIGOPS Oper. Syst. Rev.*, 42(3):83–92, 2008.

[17] B. W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, 1973.

[18] G. F. Lyon. *Nmap Network Scanning*. Insecure.Com LLC, 2008.

[19] W. Schindler. A timing attack against rsa with the chinese remainder theorem. In *CHES '00: Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*, pages 109–124, London, UK, 2000. Springer-Verlag.

[20] D. X. Song, D. Wagner, and X. Tian. Timing analysis of keystrokes and timing attacks on ssh. In *SSYM'01: Proceedings of the 10th conference on USENIX Security Symposium*, pages 25–25, Berkeley, CA, USA, 2001. USENIX Association.

[21] P. Szor. *The Art of Computer Virus Research and Defense*. Addison-Wesley Professional, 2005.

[22] J. C. Wray. An analysis of covert timing channels. In *IEEE Symposium on Security and Privacy*, pages 2–7, 1991.