At midnight your phone goes off at a rate you’re certain means the end of the world. Rubbing the sleep from your eyes you open it up, and view the first message. To your horror, there are hundreds of pages for network cyber incidents. You rush to the computer only to find tens of thousands of Intrusion Detection System (IDS) alerts, but there’s one large problem, every alert triggered is for devices that don’t exist on your network. There are no devices in the network that correlate to any of the devices within those alerts. Is it possible your IDS is busted? Are you actually under attack?

I am compelled to share with you a groundbreaking approach to evaluating an IDS, one which can put the operational state of an IDS’s to question. In a landscape where going quiet can be how you get caught in an enterprise environment, using the tool aptly named Pandemonium brings a new approach. This tool harnesses the power of deception, to simulate a scenario where all network threats an IDS detects become active simultaneously. Its utilization, able to demolish the confidence of any response team in their IDS.

I am Harrison Koll AKA SnipSnapp, a security engineer who enjoys building cybersecurity tools, and a deep-seeded believer that good network security forms the bedrock of any halfway decent security solution. Working as a security engineer for a security operations center (SOC), I am no stranger to responding to potential security incidents. After working such a job, I recently had to perform multiple proof of value (POV) assessments comparing the operational technology (OT) IDS’s of Microsoft, Palo Alto, and Claroty. During my extensive assessment, I found there was no great way to quantitatively evaluate each of the IDS’s against existing solutions. To evaluate each of these tools I would’ve either had to put the already vulnerable OT systems at risk by launching attacks on systems to evaluate for detection, or find and replay packet captures. Both of these approaches taking prohibitively long, but necessary lengths of time. As a solution to this problem Pandemonium was created; a tool which uses network signatures to build, play, and effectively replicate malicious traffic on-the-fly.

**What it does**

Pandemonium takes a network signature in the form of a Snort rule. The parameters for each snort rule are not compared to traffic, but are used as the building blocks for new, spoofed traffic. The tool dynamically packet crafts entire fake conversations between fake or real devices and plays them over the wire. In doing this, it has the ability to spoof MAC and IP addresses for any arbitrary address. Given any single rule being played, the traffic generated will be as random or as specific as the rule is and parameters given to it. It does not matter how nonspecific each rule may be. The less specific the rule, the more randomness which may be injected into each payload produced by Pandemonium. On its inception, the goal of Pandemonium was to play such traffic so defenders would be able to quantitatively compare what detections they may be losing when changing their IDS solution.

**How it works**

Pandemonium was built to do the complete opposite of what Snort does. To do this, it takes in a literal Snort rule configuration, Snort rules, and other user-configurable settings. The most important of these two files are the Snort configuration, and Snort rules. The Snort configuration file is necessary as it is used to associate all of the variables contained in the snort rule files to their values (just like Snort does). But to generate any actual payload, the snort rules are what Pandemonium uses. Every rule in the designated Snort rules files are evaluated. For instance, take this snort rule:



When playing and building traffic, PANDEMONIUM first carves out the unimportant pieces of the rule, and splits each rule out into header, and rule content portions. So that the rule becomes:



First the header is evaluated, the first parameter for this example is tcp, and TCP becomes the IP protocol that will be used for the rule. Each other singular piece is used just like snort. For instance for the next portions of the rule, $EXTERNAL\_NET and $HTTP\_PORTS will be evaluated and ultimately specify the client’s IP address, and port. Because these are Snort variables, and should be defined in a Snort configuration file, these variables will be replaced by whatever variable associated with it that can be found in a Snort configuration file. Just as with Snort, if $EXTERNAL\_NET is defined as “any” then any randomly generated IP will replace the $EXTERNAL\_NET variable. If an array of networks or IP addresses is given, then a random IP selected out of the network space will be used. The $HTTP\_PORTS will behave similarly. Replacing the port number with a random port defined from the configuration file, so if the snort configuration variable $HTTP\_PORTS is defined as such:



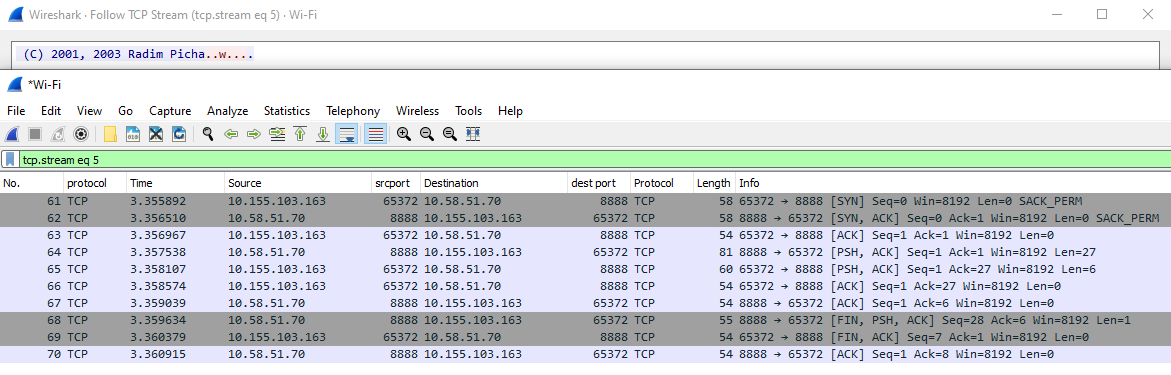
Then a random port out of the array of ports will be assigned to a random value defined from the configuration defined array. The next piece of the snort rule to be evaluated is the directionality or ‘flow’ of a snort rule. This is designated by the “->” or “<>”. Where “->” designates client to server, and a “<>” meaning a bidirectional conversation between a client and a server. This part is normally thrown out when a Snort rule is played, but is useful for evaluating the validity of a rule. Finally, the server variables are evaluated. These parts are evaluated the same as the client portion is evaluated as shown above.

Rule content is not parsed until each rule is played to help save on memory. Within each rule content, or payload portion, it is primarily evaluated as an array with the end of each element divided by a ‘;’ character. Each singular array part is then further divided up by every ‘:’ character, turning making each payload a natural dictionary of fields that can be used. So in our example, the values for the payload portion can be seen as the python dictionary:



In this rule, the “flow” parameter is used to specify whether the client, or the server will send the payload, and to decide whether or not the TCP connection will be gracefully closed. The remaining portion, the rule content is evaluated like a normal Snort rule.

When PANDEMONIUM played this particular rule, the traffic looked like this on my network in wireshark:



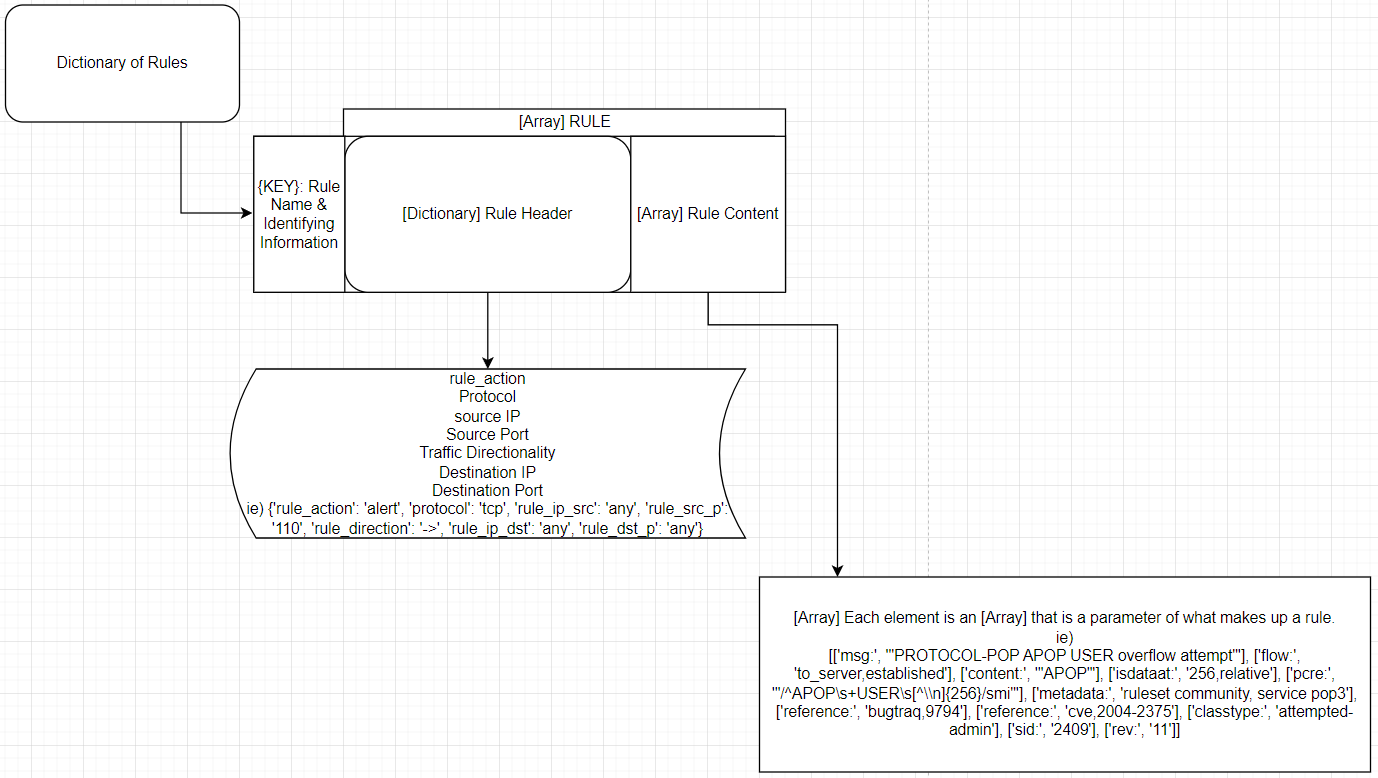
The above is one example start to finish of how a rule is processed and played in order to do so. An IDS works by observing the conversations going on in a network, and evaluating those conversations for atypical or malicious chatter.

**How Pandemonium was designed and why it was made**

In building Pandemonium I took a look at how Snort works. After exploring the backend of many Intrusion Detection Systems over time, I found most Pay-To-Play intrusion detection systems utilize snort or a variation of it. Even if an IDS supposedly uses “AI” to assist with their detections. At the end of the day, I could always find some Snort rules packed away in some SQL database, or tucked in a poorly obfuscated rules file when it came to these “AI” Tools, and most of the “AI” evaluation came to simple source traffic identification for Snort. The reason for this is Snort is well known, easy to modify, and easy to create rules for. It’s actually a great practice, innovating off of what’s already there to make it better! So knowing this, Pandemonium would do the same by leveraging Snort, making it so any security professional would be able to build and play traffic.

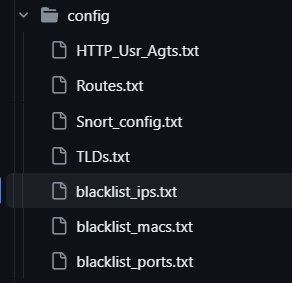
The next problem would be actually playing the traffic from a SNORT rule over the wire. In college, folks’ often mentioned packet crafting in passing. Luckily, Philippe Biondi created their own packet manipulation toolset called Scapy, which is able to craft whatever packets we please. Scapy is a beast of a tool, and Philippe really outdid themselves in making it. There are versions of it for both python and Java, it works on OSI layers 2-7, has its own native routing table. For these reasons, Scapy, and Python became core technologies to build Pandemonium with.

The next problem was transforming Snort rules so Scapy could process them, and fire off traffic. Pandemonium also needed to store these rules so they could be played once parsed. Pandemonium would also need to take snort configurations into account to make it effective. To parse rules, Pandemonium heavily relies on python cutting out the parts of a rule from rule files that are important. Pandemonium stores rules in a structure best described as a Dictionary of rules, with the keys being some identifying information for each rule being the key. The value of each key corresponds to the actual meat of each rule and is divided into two parts. The rule header which is another dictionary, which contains keys for the rule action, protocols, ports, directionality, etc. And the second element of the rule array being another array, which contains an array with all of the content for each rule (diagram below).

The resulting structure is the product of on-the-fly programming, but it works quickly in both parsing as it can dynamically accommodate thousands of rules, and dynamically playing each rule. Rule content is stored as its own array because rule content is evaluated dynamically. This non-deterministic approach was implemented for two reasons when Pandemonium was built. The first to give the tool the illusion of doing encryption, the other reason is to make some of the more high-end “AI” based tools believe active malware is afoot. 

Rule content can be as exact, or as fluid as SNORT itself allows them to be. To evaluate regex, and support it Perl is used to transform regex, and expand it into something meaningful. In the end, when PAndemonium is done with its transforms and translations of rule content, it turns into a byte array neatly packaged as a payload Scapy can understand. And finally the rule is played over the wire.

**Additional Configurations**

So that playing signatures can be as versatile as possible, PANDEMONIUM comes with a large set of configurable options; 

1. HTTP\_Usr\_Agts.txt
   1. This specifies the different user agents that are options when playing HTTP rules. To remove the various options for user agents that can be used, simply delete or add entries to this list.
2. Routes.txt
   1. Remember how SCAPY has its own routing table independent of the machine you use it on? PANDEMONIUM leverages this by allow you to define custom routes for the flow of your traffic. To use the default routes provided by your system, just delete the entries in this file.
3. Snort\_config.txt
   1. This is where you place a snort configuration. The snort configuration must reflect the snort rules that you input to be played. All of the variables that will be used inside of a rule that is played need to be in the Snort\_config.txt file. If any variables that are typically used for a snort network signature are not found within this file, you will have problems playing the different snort rules. PANDEMONIUM uses the variables for the networks, or IP addresses specified by these signatures to generate traffic. Ports used may also be defined within this file. The Snort\_config.txt file that contains a snort configuration is just as important as it is within a normal snort distribution.
4. TLDs.txt
   1. This file contains a list of top level domains (TLD’s) that are used when playing HTTP rules. To change the top-level domains that are used simply add or remove any TLD’s from the list and they won’t be used for random URL generation when generating HTTP traffic.
5. blacklist\_ips.txt
   1. A list of blacklisted IP addresses specific networks that you may want to avoid when it comes to playing traffic. If this list is left alone, when playing traffic, all internet IP addresses are avoided when traffic is played. Networks that should be avoided for playing traffic can be specified either by listing a singular IP address, or specifying the network through CIDR notation. To play traffic using a public IP address, all one has to do is delete one or more of the entries listed in the blacklist\_ips.txt file. This file defaults to all public address ranges so you don’t anger an ISP.
6. blacklist\_macs.txt
   1. A list of MAC addresses that won’t be spoofed when playing traffic. This is important to modify to make sure that you don’t enter a MAC address that can cause a switch to drop a frame. There are some group MAC addresses, and broadcast MAC addresses listed within this file to get you started but not all of them.
7. Blacklist\_ports.txt
   1. A list of ports that should not be spoofed when playing traffic. This file contains any ports that you don’t want used for spoofed traffic. This file is left blank intentionally, but was added as a precaution.

**Why You Should Care**

Incident responders rely on the integrity of alerting systems to identify and protect corporations from harm. Using Pandemonium, evaluating an IDS turns a once arduous task into a trivial one. Pandemonium can also be used for evil though, because playing spoofed traffic can be used to drop the alert integrity to zero for an IDS. every alert could be bogus. And when there are thousands of these kinds of alerts it can be used to confuse a defender. With current approaches in IDS systems to provide only the metadata of alerts and nothing meaningful, an incident responder would have a hard time separating real from fake when Pandemonium is active on a network. The ability to generate thousands of fake alerts on an IDS in minutes is a powerful thing to have.

**Download it Here:**

[**https://github.com/SnipSnapp/pandemonium**](https://github.com/SnipSnapp/pandemonium)

**POC Video Here:**

[**https://www.linkedin.com/posts/harrison-koll-3aa806189\_project-video-snort-activity-7029241166126596096-TmmD?utm\_source=share&utm\_medium=member\_desktop**](https://www.linkedin.com/posts/harrison-koll-3aa806189_project-video-snort-activity-7029241166126596096-TmmD?utm_source=share&utm_medium=member_desktop)