Challenges of fiction in network security – perspective of virtualized environments

Vit Bukac, Radim Ostadal, Petr Svenda, Tatevik Baghdasaryan and Vashek Matyas

Faculty of Informatics, Masaryk University, Brno, Czech Republic
{bukac, ostadal, xsvenda, matyas}@mail.muni.cz, tatbagg@gmail.com

Abstract. The paper aims to start a discussion about challenges and possible caveats of performing network security experiments with high traffic volumes in virtual environment. A new framework for rapid evolution of denial-of-service attacks by genetic algorithms is presented. Issues with virtual environment that were encountered during initial work with the framework are listed.

Keywords: virtualization, framework, genetic algorithm, denial of service attack

1 Background

Virtualization has a significant impact on how network security experiments are performed. It allows for a high flexibility in both experiment design and scope setting, and it also supports experiment repeatability with quick restoration of predefined state. Virtualization enables a great utilization of available resources with a high scalability. Yet are the environments that were built over standard hypervisors (e.g., Xen, VMware, Hyper-V) truly representative? Are experiment results obtained in virtual environment applicable also in physical environment?

We have designed a new generic framework for quick automated evolution of denial-of-service (DoS) attacks in virtual environment. The framework is sufficiently universal to be used for evaluation of an arbitrary denial-of-service attack. However, we have encountered issues that could have a huge impact on the results collected in similar environments. We would like to initiate a discussion about issues that could change our perception of virtual environment as a helpful servant.

It has been known for a long time that network simulations do not reflect behavior in real networks correctly, especially when considering high-volume traffic. Research on denial-of-service attacks has been badly affected, because simulations assume ideal environment without limitations of the physical world (e.g., sizes of buffers on routers) [CFS06]. As a response, emulation testbeds [MBF+10] and hybrid physical-virtual testbeds [SST+10] have been built to support experimentation with large-scale attacks.

2 Our framework

The framework has been initially created to examine possible enhancements to the HTTP GET flooding attack by modifying HTTP request headers. The project aim was to search for such HTTP GET headers where their processing by the victim server would be significantly more resource demanding than the processing of HTTP GET headers from common web browsers.

The framework applies genetic algorithms to existing DoS attacks in order to discover advanced, more potent attack variants and also to identify DoS vulnerabilities in applications that are serving as targets. The architecture is outlined in Figure 1.

The framework consists of a central management host and multiple physical computation hosts for conducting experiments themselves. Each computation host has a hypervisor installed and hosts two virtual machines attacker and victim. Each management unit can assign tasks to multiple computation hosts, therefore each generation can be evaluated on dozens of physical hosts simultaneously. Virtual machines (VMs) on different physical hosts are clones of attacker and victim initial source VM images. Thanks to snapshot restoration, each evaluation is performed in exactly the same virtual machine state.

The modular architecture enables seamless changes. Employed genetic algorithm, virtual machine operating system and target application can all be changed with minimal impact on the other parts of the framework. Once the task and its properties are fixed, the framework can be left to produce relevant results automatically.

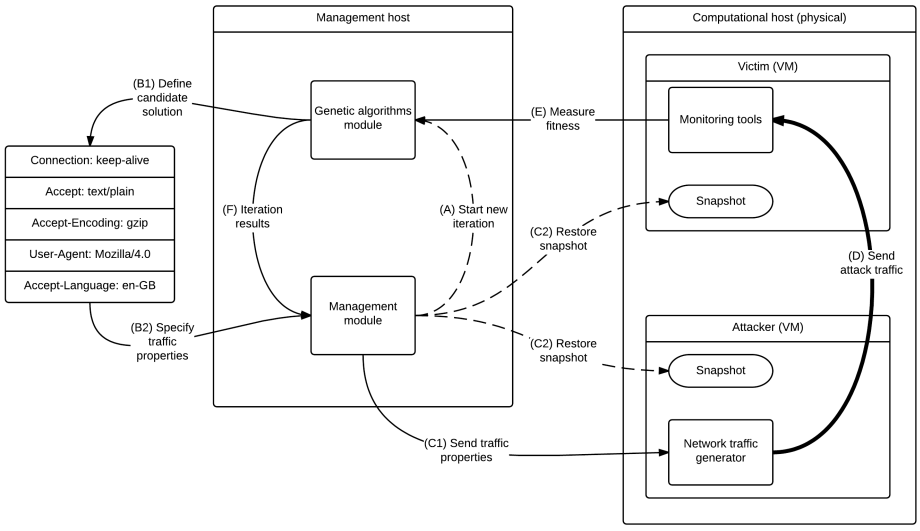

Fig. 1. Framework architecture

2.1 Workflow

A – Management module initiates evaluation of a new population.
B1, B2 – Genetic algorithms module creates a list of candidate solutions to be evaluated in the current round and provides the list to the management module. Each candidate solution provides a representation of network traffic that needs to be evaluated.
C1 – Management module maintains a list of active computational hosts. Candidate solutions are distributed equally to all available computation hosts in order to minimize the time required to evaluate the entire population.
C2 – Management module restores snapshots of all virtual machines on computation hosts. Restoring snapshots is quick and establishes a common initial state for evaluation of every candidate solution.
D – Attacker virtual machine contains a network traffic generator that can translate the received specification (i.e., candidate solution) into an arbitrary network traffic. The actual generated stream of packets is sent towards the victim virtual machine.
E – Monitoring tools on the victim VM measure the impact of received traffic on the host (e.g., consumption of RAM, CPU load or values of application-specific performance counters). Measured values are converted into a common format and sent as fitness function values (how well candidate solution satisfies the goal high load in our case) to the genetic algorithms module.
F – Genetic algorithms module evaluates all received fitness function values, chooses the best solution(s) and provides results to the management module for a manual review. Once enough results are received, management module starts a new round.

3 Scenarios

3.1 HTTP requests

As mentioned before, our original goal was just to search for such HTTP GET headers that could create a burden on the victim server with significantly more resource demanding load than would be that of a processing of HTTP GET headers from common web browsers.

A candidate solution is an ordered list of pairs (HTTP header field, value). Candidate solutions differ in the chosen header fields, appropriate values and the order of pairs in the header. Each candidate solution is incorporated into HTTP GET request with a constant URL before being sent. The URL targets a copy of a well-known news webpage which is running on the victim VM. Each request is sent 10 000 times. Monitoring tools on the victim VM collect CPU time of all Apache processes.

Measured values were afterwards compared to CPU time of HTTP requests that were constructed to mimic requests from common web browsers (i.e., Google Chrome 35, Internet Explorer 11 and Mozilla Firefox 31).

3.2 HTTP requests – Lessons learned

We have encountered a number of problems to cope with:

- **Measurement precision**. While the deviation of multiple measurements of the same phenomena was less than 3% with physical hosts, the deviation increased up to 20% when similar measurements were conducted in virtual environment.
- **No relation of results in virtual environment and physical machines**. We were unable to replicate some results from virtual environment on real hardware. The example could be the Figure 2 with results from virtual environment. In virtual environment, Best 1, 2 and 3 requests require higher CPU load than the common IE request. When we sent the same HTTP headers on two separate physical machines, the difference between them and IE was negligible.
- **Different interpretations on physical machine and in virtualization**. We observed that the same version of Wireshark on the same version of operating system interprets the same network traffic differently, when running on real HW and when running in a virtualized environment. This behavior could influence any automated analysis of PCAP files that is based on the libpcap library.
- **Incomparable performance from hosts with different hardware configuration**. Virtual machine performance is heavily influenced by underlying physical hardware. Two virtual machines running on different hardware will provide different measurement, even though the environment seems to be exactly the same when observed from inside. All candidate solutions must be evaluated on computational hosts with similar HW configurations. This presents a significant challenge for comparability of any cloud-based computations.
- **Cable vs. Wi-Fi connection**. We also applied a variant where attacker VM and victim VM resided on separate physical hosts. Operating system performance counters values were distinctively different when attack traffic was sent through Wi-Fi and through cable connection.
- **Lower precision bound**. When using virtualized environment, there is always background noise (e.g., fluctuations of CPU load, OS native network traffic, RAM consumption varying in time). This noise sets a lower bound for useable precision of measurements. With less than 1000 HTTP requests during each run, the noise was too dominant for measurements to have any real informational value. We therefore used at least 10 000 HTTP requests. Noise in physical environment is arguably lower.
- **Results interpretation**. Sometimes it was difficult to identify what parameters were key influencers of final results (e.g., VM configuration, physical host properties, network configuration, and internal application configuration). We had to employ try-error approach to interpret some of the observed anomalies. Also, it was helpful to collect fitness values for minimal size HTTP requests and then use these fitness values as guidance for mutual comparison of more complex headers.

Figure 2 illustrates some of our findings. Random HTTP headers represent distribution of CPU consumption of 200 randomly constructed HTTP request headers. Cluster between 0 and 10 represents malformed requests that are responded with 400 error code. Cluster between 45 and 60 represents standard common requests. Best1, Best2 and Best3 show consistency of measurements of 200 iterations of 3 most demanding requests that we were able to construct. Measurement precision is sufficient even for a fine-grained evolution. IE baseline represents consistency of measurements of 200 iterations from common Internet Explorer 11 request.

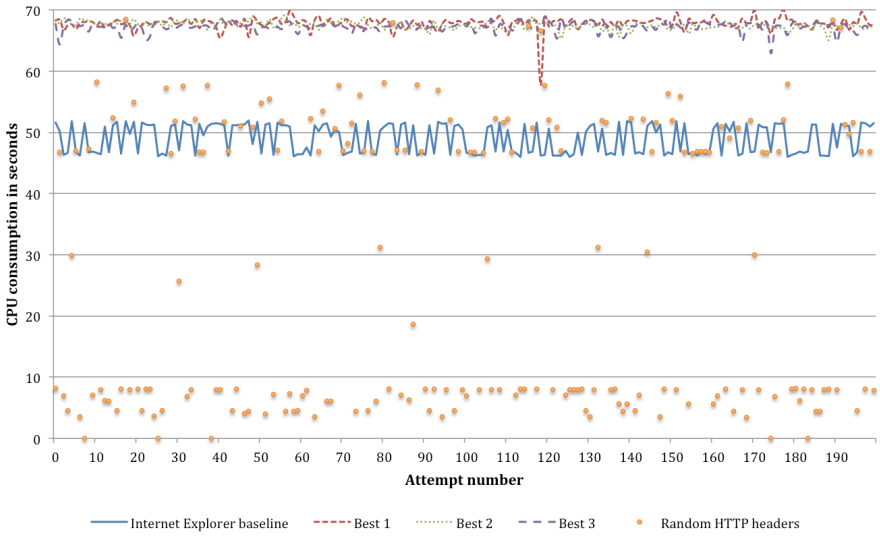

Fig. 2. HTTP requests CPU requirements

Our HTTP request project was eventually cancelled. Contrary to our hypothesis, we were unable to find a sequence of HTTP header fields and respective values whose CPU requirements would be significantly higher than computational requirements of standard browser requests. Apparently, the impact of HTTP header fields processing on a standard Apache webserver is negligible, with the exception of Accept-Encoding field. Accept-Encoding field value can significantly increase CPU consumption when zip compression is required. However, such behavior is default for common HTTP requests.

3.3 Slow attacks

Although HTTP requests research project was ultimately unsuccessful, the framework proved to be both simple and effective. Currently we are adapting it for searching for slow DoS attack opportunities in common protocols. An inherent

property of most network protocols is to proceed with next phase of protocol only when previous phase was completed. Meanwhile, each side has to allocate its computational resources for any (half-)open connection. Under normal circumstances, connections are closed only when they are no longer used, either explicitely with a close message or when an inactivity timeout expires.

For example, a webserver can only send response when full HTTP request has been received. The Slowloris attack exploits this behavior by sending a never-ending HTTP header. Therefore, the request is never finished and connection socket is effectively and indefinitely blocked for other users. If the attacker is able to maintain sufficient number of simultaneously opened connections, legitimate users cannot reach the webserver.

We intend to use our framework to identify:

- Time points of message exchange where artificial delays can be introduced into the communication.
- Separation points in each message where the message can be divided in two or more smaller messages (i.e., packet fragmentation).

Evaluation criterion will be the maximum time how long it takes to complete a given sequence of messages (e.g., how long it takes to finish a SSL handshake).

4 Open questions

We are looking for inputs and both good and bad experiences in the following areas:

- What are other limitations of virtual environment and what are the differences between virtual environment and physical hosts? How to get consistent results from virtual environment running on different hardware?
- What protocols and applications may be interesting from the viewpoint of slow DoS attack verification?
- Where to get an extensive list of available HTTP request header key-value pairs?
- Are there any similar network security evolution frameworks? Is it possible to modify existing generic fuzzers such as Peach [Edd11] to support genetic algorithms and (D)DoS principles?

5 Summary

We have designed a framework that can be used both for automated enhancement of existing denial of service attacks and for generating new types of DoS attacks. The framework shows potentially serious discrepancies between virtual and physical environments. We initiate a discussion on hidden caveats of experimenting in virtual environment testbeds.

References

[CFS06] Roman Chertov, Sonia Fahmy, and Ness B. Shroff. Emulation versus Simulation: A Case Study TCP-Targeted Denial of Service Attacks. In *2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM 2006)*, pages 316–325, 2006.

[Edd11] Michael Eddington. Peach fuzzing platform. 2011. http://peachfuzzer.com.

[MBF+10] Jelena Mirkovic, Terry V. Benzel, Ted Faber, Robert Braden, John T. Wroclawski, and Stephen Schwab. The DETER project: Advancing the science of cyber security experimentation and test. In *In Technologies for Homeland Security (HST), 2010 IEEE International Conference on*, page 7, 2010.

[SST+10] Desmond Schmidt, Suriadi Suriadi, Alan Tickle, Andrew Clark, George Mohay, Ejaz Ahmed, and James Mackie. A distributed denial of service testbed. In *What Kind of Information Society? Governance, Virtuality, Surveillance, Sustainability, Resilience*, pages 338–349. Springer, 2010.