A Bad IDEa: Weaponizing uncontrolled online-IDEs in availability attacks

Srinivasa, Shreyas; Georgoulias, Dimitrios; Pedersen, Jens Myrup; Vasilomanolakis, Emmanouil

Published in:
IEEE European Symposium on Security and Privacy, Workshop on Attackers and Cyber-Crime Operations

DOI (link to publication from Publisher):
10.1109/EuroSPW55150.2022.00015

Publication date:
2022

Document Version
Accepted author manuscript, peer reviewed version

A Bad IDEa: Weaponizing uncontrolled online-IDEs in availability attacks

Shreyas Srinivasa[§], Dimitrios Georgoulias[§], Jens Myrup Pedersen and Emmanouil Vasilomanolakis
Aalborg University, Copenhagen, Denmark

***Abstract*—Botnets are an ongoing threat to the cyber world and can be utilized to carry out DDoS attacks of high magnitude. From the botmaster's perspective, there is a constant need for deploying more effective botnets and discovering new ways to bolster their bot ranks. Integrated Development Environments (IDEs) have been essential for software developers to write and compile source code. The increasing need for remote work and collaborative workspaces have led to the IDE-as-a-service paradigm that offers online code editing and compilation with multiple language support. In this paper, we show that a multitude of online IDEs do not run control checks on the user code and can be therefore leveraged by a botnet. We examine the concept of uncontrolled execution environments and present a proof of concept to show how uncontrolled online-IDEs can be weaponized to perform large-scale attacks by a botnet. Overall, we detect a total of 719 online-IDEs with uncontrolled execution environments and limited sandboxing. Lastly, as ethical disclosure, we inform the IDE developers and service providers of the vulnerabilities and propose countermeasures.**

1. Introduction

Botnets malicious networks of infected systems responsible for high-impact cyber attacks like Distributed Denial of Service (DDoS). They operate by leveraging vulnerable devices connected to the Internet to execute attacks and are managed through a command and control system (C&C Server). Mirai-like malware has caused high-impact attacks in the past, and infected vulnerable IoT devices for performing DoS-like attacks [1]. Moreover, research studies indicate that browser-based bots are more effective and economical than malware-based bots [2]. The ENISA Threat Landscape Report 2021 states a rise in newer malware used for Denial of Service attacks, ransomware injection, and crypto mining. The report further states that the number of attacks due to malware is decreasing from previous years; however, the focus of newer malware is reduced on quantity but possesses increased quality [3]. This entails that bot developers are exploring newer delivery methods that are more discreet and effective [4].

Programmers have traditionally used IDEs (Integrated Development Environment) for software development. IDEs facilitate the compilation, debugging, and execution of language-specific source code. The leap in cloud computing has aided the idea of online-IDE, and REPL environments (read, eval, print, loop), that are offered as a service on the Internet [5]. Compared to traditional local-IDEs, online-IDEs have no prerequisites like installation or setup. Like local-IDEs, online-IDEs provide features like project creation, sharing, and version control that further facilitate collaboration, remote work, training, and interviewing possibilities. Moreover, many online-IDEs offer multi-language support that includes diverse programming and scripting styles. Online-IDEs are now popular for many online learning platforms, collaborative development services, and cloud-ready deployment providers.

However, upon careful analysis of online-IDEs and REPL platforms on the Internet, we observe that many do not perform checks on the user code and can be therefore leveraged to execute arbitrary code with malicious intent. Furthermore, recent research shows deceptive source-code attacks that appear different to the compiler and human eye, can be used to deceive the compilers into performing malicious operations [6]. In this work, we aim at checking the uncontrolled behavior of IDEs by executing code that leads to flooding requests on a target hosted in our lab infrastructure. Furthermore, we implement a bot that can perform Denial of Services attacks by exploiting several such online-IDE environments. To the best of our knowledge, there is no previous work which looks at leveraging uncontrolled online IDE environments to perform attacks on the Internet. Our contributions are summarized as follows:

- We examine the concept and criteria for uncontrolled execution environments and find vulnerable online-IDEs by searching the Internet.
- As a proof of concept, we implement a bot that exploits the uncontrolled IDEs and performs a flooding attack against a web server hosted at our lab.
- We estimate the magnitude of the attacks possible from uncontrolled online IDEs by performing multiple attack types.

The rest of the paper is structured as follows. In section 2 we discuss the background and related work. Section 3 gives an overview of online-IDEs and uncontrolled execution environments. We discuss our methodology in section 4 and section 5 provides an evaluation of our approach. In section 6 we discuss the attack types and the limitations. Section 7 describes the ethical considerations followed in our methodology and disclosure. We discuss the future work and conclude in section 8.

§. Equal contribution

2. Background & Related Work

2.1. Online IDEs

Modern IDEs provide features that help in accelerating development with the use of Artificial Intelligence, collaborative development, cloud deployments [7], and additional features that include build automation tools, class browsers, object browsers, and version control. Online IDEs provide most features of a local IDE with the advantage of no installation required on the user system and with the possibility of remote access. However, there are some issues specific to online-IDEs, like running static and dynamic program analysis on the user program [8]. The rise in demand for online platforms and cloud deployments, leads to an increase in online IDE services offered for training, assessments, and development environments. With this increase, there is also a risk of potential exploitation of these platforms, wherein an adversary could use them to spoof the attack source. Adversaries and botnet campaigns can utilize uncontrolled online-IDEs to cause varied attacks such as availability attacks, crypto mining, and malware injection [9]. For example, *PyCryptoMiner*, a Linux crypto-miner botnet, spreads through a compromised SSH service and deploys a base64-encoded Python script that connects to the command and control (C&C) server to fetch and execute the crypto-mining Python code [10]. The bot mines the Monero [11] crypto-currency, which is the preferred mode of payment in the Darkweb [12]. Furthermore, botnets like Mirai have caused availability attacks of high magnitude by infecting vulnerable systems [1].

2.2. Uncontrolled execution environment

Although there is research towards enhancing the capabilities of IDEs through the inclusion of static and dynamic testing plugins, there is little work on control of execution in online-IDEs. Wu et al. summarize the problems in online-IDEs in regards to uncontrolled execution into (i) wrong file operations, (ii) banned method calls, and (iii) excessive resource consumption that can lead to arbitrary code execution and resource depletion [13]. The authors indicate that IDEs must offer partial file-based operations; for example, deletion of a file on the platform must not be permitted. Similarly, the authors specify the need for banning specific methods and packages that facilitate a compromise of the IDE infrastructure or remote systems. The authors emphasize the need for timeouts that limit resource consumption for the user program and present techniques that can be used to handle the three risks using a program behavior analysis and control model. The model includes static and dynamic analysis techniques to analyze the program behavior and control the code execution.

Arbitrary code execution (ACE) is an adversarial technique where the attacker can execute malicious code on the target system [14]. The attackers leverage the vulnerabilities in the target system to gain access to an execution environment. ACE is not uncommon in online-IDEs as they offer users an open code execution environment. While the main objective of these IDEs is to provide an online execution environment, there is little focus on controlling the environment for any malicious code (for example, HTTP Flood requests). Kiransky et al. propose program shepherding, a method for monitoring control flow transfers during program execution to enforce a security policy [14]. The authors provide three techniques from program shepherding that act as building blocks for security policies, including restricting execution privileges, restricting control transfers, and sandboxing checks. The authors further present a detailed approach to security policies regarding program shepherding that ensures malicious code detection through multiple techniques and prevents execution. Program shepherding includes IDE sandboxing, ensuring that malicious code execution does not impact external systems.

The approach closest to our work is from Pellegrino et al. [2], where the authors explore the idea of using browser-based DDoS botnets and review ways attackers can weaponize them. The authors present three ways of using browser-based JavaScript to initiate thousands of HTTP requests per second and evaluate the costs compared to a traditional botnet. In our work, we specifically search the Internet to find many online-IDEs that do not have controlled execution environments and have limited sandboxing capability. We demonstrate the impact through various attack types originating from these vulnerable IDE instances. To the best of our knowledge, our work is the first to explore the area of uncontrolled execution in online-IDEs and assess the impact of potential attacks through measurement and estimation.

3. Uncontrolled execution environments

This section describes the generic architecture of online-IDEs and the criteria for classifying the online-IDEs into uncontrolled execution environments.

3.1. Generic architecture of online-IDEs

The architecture of online-IDEs can be broken down into three components (i) frontend, (ii) backend, and (iii) messaging service. Figure 1 shows the generic architecture of online-IDEs. The frontend component provides a GUI as a web application for the user to input the source code, execute button, and a window to view the output of the executed code, similar to that of local IDE. The backend component contains the compiler and the file system that compiles the user code. This component can either share the host of the frontend or on a remote host for scalability purposes. The messaging service is responsible for transporting the user input (source code) from the frontend to the backend and the output from the backend to the frontend. In addition to the frontend, backend, and message transport, some online-IDEs offer extensions that provide collaboration, version control, build tools, and other features. There exist opensource online-IDE frameworks such as Eclipse Theia [5], ICEcoder [15], Microsoft VSCode [16], Code-server [17] and AtheosIDE [18] that follow similar architecture. However, we observe from our reconnaissance that most online-IDEs have their stack similar to the generic architecture.

3.2. Uncontrolled online-IDE environments

In order to classify an online-IDE as having an uncontrolled execution environment that can be exploited and

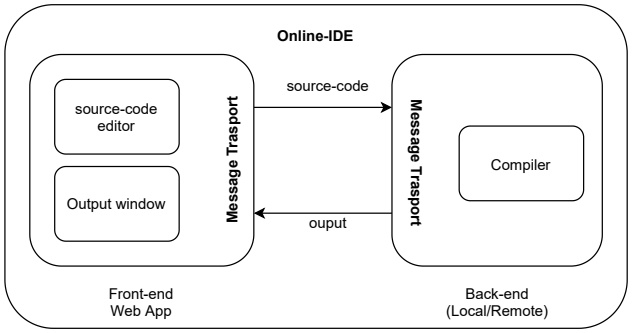

Figure 1. Generic architecture of online IDEs

weaponized to carry out flooding attacks, we adopt and extend the criteria defined by Wu et al. [13].

3.2.1. Unrestricted file operations. File operations such as read, write and modify are facilitated through the default packages in many programming languages. The file operations facilitate data import from files and export of output. Most online-IDEs support the import of source-code through files, which we observe are not validated for malware. Furthermore, it is crucial to restrict access to the file system on the hosting system. Unrestricted file operations on the online-IDEs can lead to malware injection, installation of malicious libraries, and corruption of the file system that can compromise the availability of the service.

3.2.2. Unrestricted package/module import. Programming languages depend on modules or packages for specific operations like creating HTTP requests or file handling. Developers import/include these packages into their source code to support such operations. We observe that an adversary can leverage unrestricted package imports for malicious purposes like creating HTTP floods or malware injection.

3.2.3. Unbounded resource consumption. It is crucial to prevent or limit the execution of a program that consumes resources as this may entail resource depletion and eventually cause the online-IDE to crash. However, many online-IDEs run on scalable cloud platforms. Bots can leverage such online-IDEs to carry out high magnitude attacks or inject crypto-mining malware. An adversary can further leverage the elastic resources offered by the IDE to run malicious code that exploits remote systems.

3.2.4. Non-sandboxed environments. Sandboxing is a mechanism in which an instance is isolated to prevent any spread of vulnerabilities or infections to other machines in the network. Furthermore, sandboxed environments offer controlled use of underlying resources and are ideal for executing untested code. Non-sandboxed environments are risky, allow for access to networked systems, and can be used to exploit remote systems. Non-sandboxed online-IDEs in particular are ideal for malware spreading. Botnets can inject malware or execute malicious code to cause a flood on remote systems and further allow communication between the infected system and the control server.

3.2.5. Stateless runtime sessions. Web servers maintain sessions to maintain the current user's data for a period. Online-IDEs can use sessions to track the use and the requests from the current user to limit them to a specific period. Moreover, online-IDEs can use sessions to track the user's state and stop the program execution when the user state is idle or disconnected. In stateless sessions, the online-IDEs do not keep track of the user sessions, which can be exploited by a user running arbitrary code and terminating the session while the online-IDE is still executing the user code. The user can create multiple sessions, run arbitrary code and close the sessions while saving system resources. Furthermore, an online IDE must also restrict the number of sessions per user for controlled resource usage.

4. Methodology

This section presents our methodology for finding online IDEs, checking for uncontrolled execution, and leveraging them in our botnet for performing a Denial of Service attack.

4.1. Reconnaissance

We use Google Dorking to find IDEs on the Internet using. Google Dorking is the process of finding specific web pages on the Internet by using search parameters with keywords [19]. For example, *intext:"online IDE"* returns a list of online IDEs. The search parameters can be narrowed to find language-specific IDEs, like *intext:"online python compiler"*. The keyword search in the dork process of our approach has specific keywords, for example, "Python Online IDE" that provide language-specific results. However, some online-IDEs support execution of multiple languages. In this case, we manually determine the languages supported by the IDEs to check for multi-language support. This work limits the proof of concept to include online-IDEs that support Python language execution. Moreover, we leverage datasets from Internet-wide scanning services like Censys [20] and Shodan [21] for searching for online-IDE instances using keywords like "online IDE", "online REPL", and further filtering the results using common labels contained in the HTML of code-editors like the Ace editor [22] (e.g. JavaScript editor syntax like *ace.edit*).

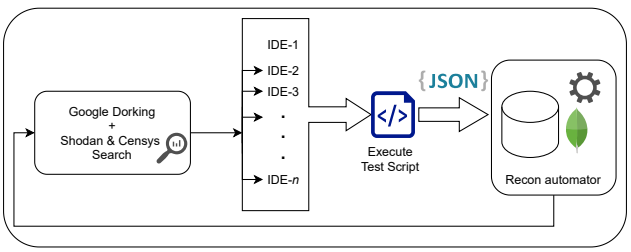

Figure 2. Reconnaissance-phase automation

The overall reconnaissance process is illustrated in Figure 2. After the Google Dorking process, we use language-specific scripts to check if the IDEs from the search results support uncontrolled code execution and

group them based on language in our database. The IDEs are further checked for uncontrolled execution parameters by execution of custom language-specific scripts. The output of the scripts is posted to a remote server repository as a *json* document embedded in an HTTP post request. The server repository contains a list of all the IDEs with an uncontrolled execution environment, the language supported by the IDE, and additional metadata about rate limiting. The reconnaissance process is performed weekly to find new IDE environments and check if the existing IDEs in the list are still unpatched.

To check for the uncontrolled execution parameters defined in Section 3.2 we use the following approaches in our proof of concept script:

- Unrestricted-file operations: To check for restricted file read and write operations, we try accessing the environment variables from the host. The access to environment variables further helps determine if the IDE is operating in container mode. We use the checks os.environget() to read the HOME variable and os.environ['FOO']='1' to create a new variable in our proof of concept script. To check for delete operations, we remove the environment variable created during the test write operation. We set flags to determine the successful operation. In addition to the above checks, we check for access to some critical file system paths.
- Unrestricted package/module import: We import packages from python default libraries such as sockets, threading, and os that can be used for our proof of concept. The successful import of the packages is determined by implementing checks in the proof of concept script.
- Unbounded resource consumption: By importing the threading package from the previous check, we implement a function that can create a large number of HTTP requests over time. In this way, we check for both unbounded resource consumption and network rate limiting.
- Non-sandboxed environments: To check if the python code is executing under a container mode, we read the "/proc/self/cgroup" and check if the field "docker" exists. In addition to the container check, we check for Internet access through the IDE host by executing a simple HTTP request through the sockets package.
- Stateless runtime-sessions - To check if an online-IDE environment supports stateless runtime sessions, we execute multiple HTTP requests to our test webserver for a specific period (i.e., 5 seconds) and close the IDE session. We measure the number of requests received and the period to check the total execution time from the first request received.

4.2. Botnet architecture

In order to explore the magnitude of the discovered vulnerability, we decided to develop an application that would, to some extent, simulate the operation of a real-world bot, part of a botnet architecture. This translates into the botmaster having control over the bots and utilizing them to carry out attacks at will. Additionally, in this particular case, the botmaster does not have to worry about propagating the bot malware to infect new hosts and increase the size of their network, and also the availability of the bots/IDE instances is very high since they are running on active websites.

Overall, there are only two requirements for the botnet to be functional, namely discovering the IDEs (see Section 4.1) and then including them in the *botmaster application*.

4.2.1. Botmaster application. The basis for the vulnerability's exploitation lies within the arbitrary execution of code on the IDEs. In our implementation, which was developed in *Java*, for this task we use the *Selenium* web browser automation software and a *Chrome WebDriver*. Locating the *XPaths* of the elements of interest in the HTML of each website allowed for the interaction through Selenium, which runs locally in our code editor. The most vital elements in common in all IDEs were the code editor text-area, the programming language option, and the run/execute button. These XPaths were hardcoded into the application, and since they differed in each platform, the entire process had to be repeated uniquely for each IDE. Lastly, to provide ease and simplicity to our experiments, as part of the botmaster application, we included an interface that can be used to navigate through the list of available IDEs and to orchestrate attacks by "tailoring" the *Bot Attack Code* (see Section 4.2.2) which can be deployed on the IDE, through a *Start Attack* button.

4.2.2. Bot attack code. The bot attack code is an HTTP flood attack; it was found publicly available on a *GitHub* repository [1] and is a part of the botmaster application. It is written in *Python*, and it was slightly altered in order to match the requirements of our experiments. The attack revolves around 4 discrete variables, the *target*, the *attack duration*, the *number of utilized IDEs*, and the *sessions per IDE*. After inserting the values of these variables in the botmaster interface, the attack code runs on the chosen IDEs. At this point, we noticed that a large number of IDEs would have an issue providing the expected indentation in the code using the carriage return $\backslash r$ and new line $\backslash n$ whitespace characters. Hence, the solution was to create two separate attack classes, one that would use the whitespace characters and one that would use Selenium keyboard commands such as ENTER, HOME, and BACKSPACE, to achieve the desired outcome.

4.3. Experimental setup

We leverage 18 online-IDEs supporting the Python program execution discovered during the reconnaissance process. To ethically involve these IDEs in our experiment, we ask for consent from the environments that provide contact information about the owner. The experimental setup is described in Figure 3. We deploy an *Nginx* web server in our lab to target the attacks from the IDEs. We set up the *Zeek-IDS* on the host of the webserver to log all the ingress traffic on the webserver.

To visualize the logs from the Zeek-IDS and monitor the resources on the host, we use the *Kibana* visualization

1. https://anonymous.4open.science/r/Bad-IDEa-1078

dashboard. All ingress traffic towards the server is logged. The web server and Zeek are set up on a host with a quad-core Intel Xeon processor and a memory of 32 gigabytes. The Nginx web server listens on the HTTP port 4444 and is deployed with the default configuration. The Elastic search database and the Kibana dashboard run on a remote host, and the Zeek logs are shipped using an Elastic agent. The bot developed for the proof of concept runs on a remote client in our lab. The bot client has a quad-core Intel Xeon processor of 2.4 GHz and 32 gigabytes of memory. The web server is publicly accessible via the Internet through an unfiltered network in the lab environment.

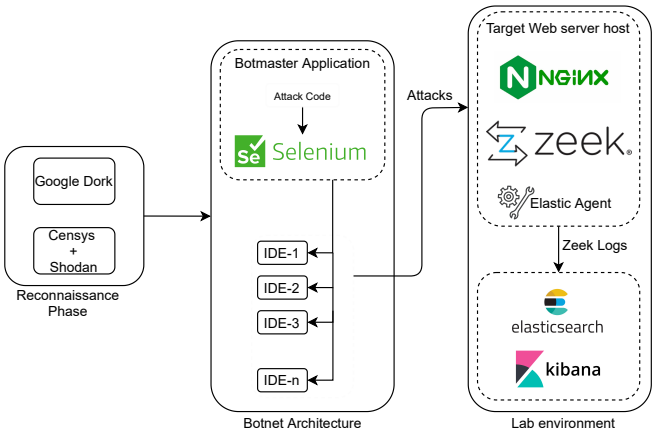

Figure 3. Methodology overview

4.4. Exploitation

Before we use the uncontrolled online-IDEs in our experiment, we ethically disclose the vulnerabilities to the service owners and developers. Furthermore, we ask for consent from the owners to use the IDEs in our experiment and test our bot. We request consent for testing from 50, but end up having consent from 18 IDE owners and test the exploit by performing an HTTP-flood for five seconds from each online IDE. Furthermore, we create multiple instances of the online-IDEs to increase the magnitude of requests. All traffic to the webserver is logged and monitored with Zeek IDS. We limit the exploitation to IDEs that support the Python language. The IDEs are listed in the bot as described in section 4.2 and the HTTP flood code that targets our web server is executed. All incoming traffic is measured per IDE and time. Evaluating the maximum traffic capability of these individual IDEs is ethically challenging without compromising the availability of the underlying host and the network. Therefore, we perform a controlled execution of the experiments to ensure that the host's availability and the network are not compromised. Furthermore, we evaluate our approach and estimate the impact of the attacks.

5. Evaluation

5.1. Reconnaissance

This section summarizes our findings from the search for uncontrolled online-IDEs.

5.1.1. IDEs found. We search the Internet through our reconnaissance approach specified in Section 4.1 to find a total of 2269 online-IDEs of which 719 had uncontrolled execution. Most of the IDEs from the results supported more than one programming language. Figure 4 shows the total number of IDEs classified based on their language support. As mentioned in the methodology, the IDEs were found through the reconnaissance process. We also observed that most of the IDEs used multiple hosts for their backend based on the language chosen by the user, and some of them did not have any login or authentication from the user before program execution.

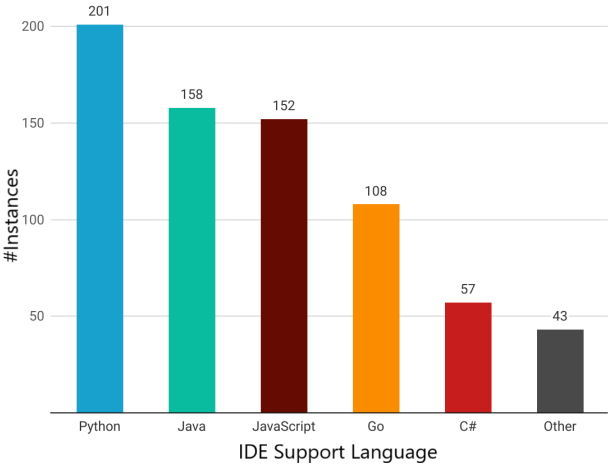

Figure 4. Classification by language support

5.1.2. Classification by use-type. Most of the online IDEs are development purpose-driven, where the user can set up a collaborative development workspace. We further classify the uncontrolled online-IDEs that we find in our reconnaissance based on their user type into an interview (24%), skill-training (22%), practice (23%), and collaborative development (31%) environments. We also find online-IDEs used as notebooks, where the user has an interactive environment with the possibility of importing datasets. Lastly, we find IDEs used by educational and training platforms that offer programming courses as a service.

5.1.3. Classification by uncontrolled-criteria. We define criteria for uncontrolled online-IDE in section 3.2 and classify the IDEs found during the reconnaissance phase. Figure 5 shows the percentage of IDEs classified based on the criteria of uncontrolled execution of online-IDEs. We observe that most of the IDEs run on non-sandboxed environments, followed by unrestricted file operations and package imports. Furthermore, we find that 719 online-IDEs from the total of 2269 from our reconnaissance

process satisfy all the criteria for uncontrolled execution. We consider this a base for evaluating our experiment further on uncontrolled online-IDEs in availability attacks.

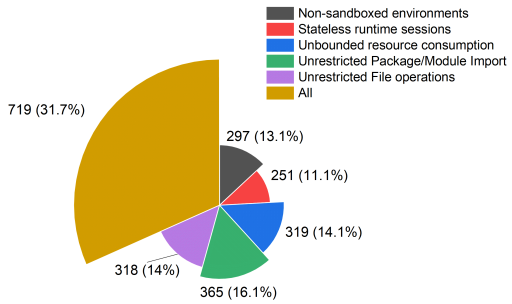

Figure 5. Classification by criteria for uncontrolled environment

5.2. Attacks & impact

We evaluate the possibility of leveraging the uncontrolled environments by performing controlled flood requests through the online-IDEs to target the webserver hosted at our lab facility. We used 18 of the total uncontrolled IDEs detected from our reconnaissance and performed code injection through the bot explained in section 4.2. The online-IDE environments that were used in our experiment did not have any user authentication or registration. Although we wanted to use as many IDEs identified in our search, we limited the number based on the consent we received for experimentation and did not cause any compromise in the availability of intermediary networking systems. Furthermore, we identified a range of 17 instances of uncontrolled online-IDEs from a reputed database provider during the reconnaissance process. As the number of instances was high, we immediately contacted the service owners about the potential misuse. Similarly, we disclosed the vulnerability to many critical operators so that the systems could be patched as quickly as possible and could not be used in our evaluation. In the following sections, we summarize the estimation, attacks, and impact of the requests sent from the uncontrolled IDEs.

5.2.1. Estimation. Performing DDoS attacks ethically over the Internet is challenging. To address this challenge, we follow an estimation-based approach to determine the impact of the attacks sourced from the IDEs. While there is existing work on mathematical modeling of DDoS attacks to predict the probability of resource depletion and bandwidth, to estimate the impact of the attacks from the IDEs, we refer to the method proposed by Balarezo et al. [23] for traffic-based models and specifically the *Queuing Model*. The *Queuing Model* uses a multidimensional approach that provides the probabilities for bandwidth, CPU, and memory exhaustion based on how networking elements process traffic. The ingress traffic measurements are carried out at periodic intervals of five seconds, and the values for the bandwidth, CPU, and memory are noted.

With the aim of developing a formula able to estimate the average attack magnitude of the architecture described in Section 4.2, we performed controlled HTTP-flood requests from the IDEs to our web server with a duration of 5 seconds, to avoid any potential disruption of the service. The experiment resulted in an average of 103 requests per IDE session, throughout all of the 32 IDEs, which proved vital in the formulation process. Figure 6 shows the average number of requests from a single session of an IDE over time up to 60 seconds, calculated using Formula 1. The figure also represents the estimated average number of requests possible from two *(n=2)* instances of an IDE running in parallel.

The variables taken into account when estimating the average total number of requests that can be achieved over a specific time interval are the number of *IDEs* used in the attack (*I*), the number of *sessions* per IDEs (*S*), the total *duration* of the attack in seconds (*D*), and the number of average *requests per second* for each IDE session (*r*). Combining all of these variables, we developed Formula 1:

$$R_{Avg} = I * S * D * r \tag{1}$$

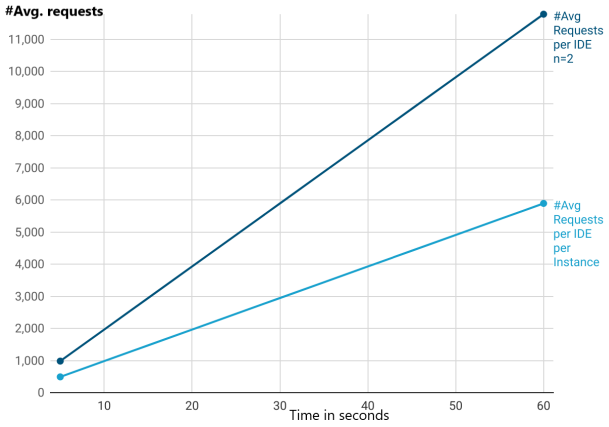

Figure 6. Estimated average requests per IDE instance by second

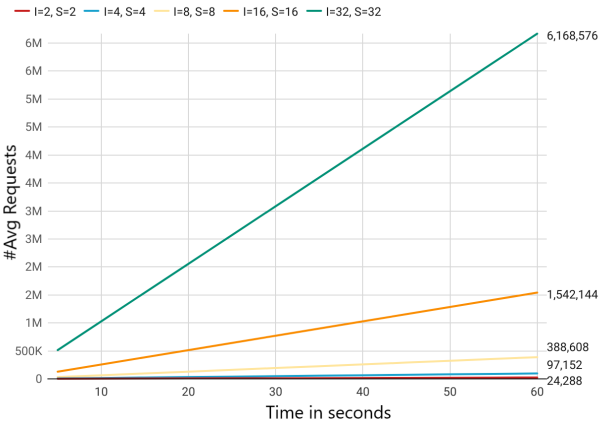

Figure 7. Estimated average requests received from multiple IDEs (I) and sessions (S)

5.2.2. Attack requests received on multiple IDE instances over time. We further estimate the number of requests possible from multiple IDEs with multiple instances running in parallel. Figure 7 depicts the estimated

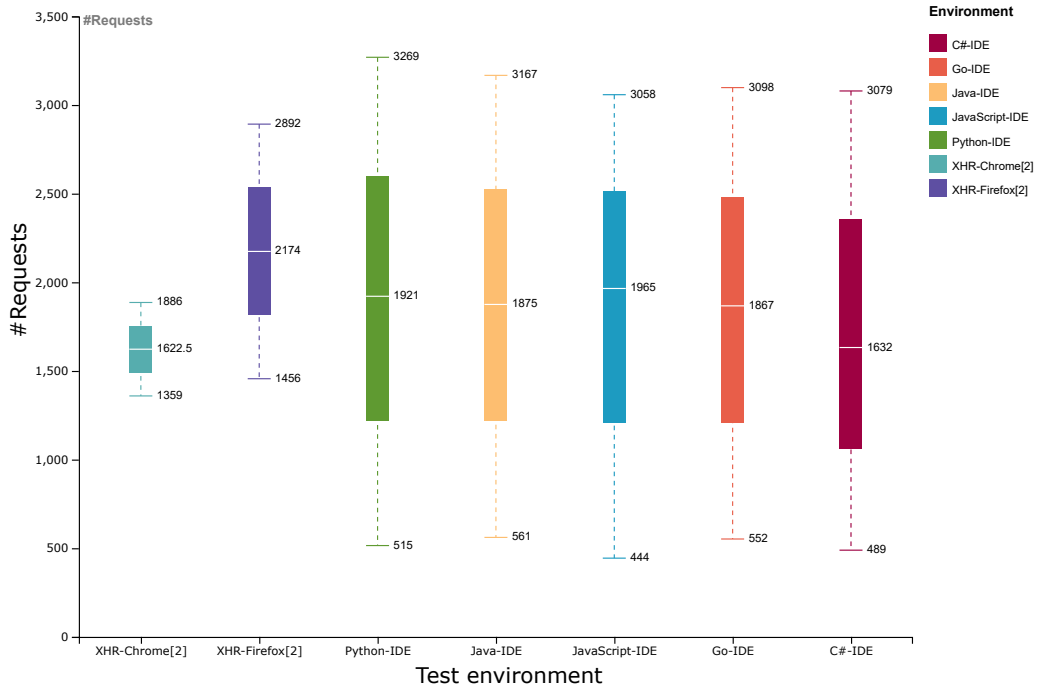

Figure 8. Average number of requests received from IDE-environments and a comparison with [2]

average number of requests from multiple IDEs denoted by I and the number of instances of each IDE denoted by S. The experiment is performed using 18 IDEs from which we received consent. We estimate an average of 6 million requests possible with 32 IDEs with 32 instances over a minute using Formula 1. The program that performs the HTTP floods is controlled by the number of threads performing the requests. We limit the number of threads to avert resource exhaustion on the IDEs.

5.2.3. Language-specific comparison. We compare the number of requests received from different languages supported by IDEs. The number of requests are obtained based on similar experiments that we carry out on Python-based IDEs. For the other languages, we perform the experiment with 6 multi-language IDEs that we received consent for experimentation. Figure 8 shows the number of requests received from different language supporting IDEs. We observe the highest number of requests from the Python supporting IDEs, in comparison with the other languages. To get a better understanding of the number of requests, we place the number of requests received from the Javascript program by Pellegrino et al. in the figure [2]. Note that this is not a direct comparison as the number of requests from the IDEs in our experiments were carried out for a period of 5 seconds and the method from Pellegrino et al. was recorded per second. However, we believe that by increasing the number of threads in our program can lead to similar results. In terms of economics, Pellegrino et al. use advertisements as a medium for executing the malicious embedded JavaScript on clicks, and hence incurs some costs. In our approach, we leverage accessible, open IDE execution environments with higher resources and negligible costs (zero) to execute the attacks.

Note that while in this evaluation, we evaluate the possibility of using IDEs that support the *Python* language, it is possible to achieve a higher number of requests by combining multiple IDEs that support other languages. While large botnets targeting DDoS attacks like the Meris Botnet have a significantly higher number of requests per second in comparison to our experiment, we believe that bots could employ vulnerable IDE instances armed with diverse attack types to increase the attack magnitude [24]. We further discuss the attack types and the impacts in the following section.

6. Discussion

Uncontrolled-IDE environments provide a degree of flexibility where the users can try performing varied attack types. Our experiments reveal that unfiltered networks of the IDEs allow different attack types. This section discusses some of the attack types that we try and describe the results.

6.1. Attack types

6.1.1. HTTP-Flood. We first evaluate our approach with HTTP-flood attacks from the IDEs. We execute the *Single Session HTTP Flood* to send a large number of requests from limited HTTP sessions. We observe that the CPU and memory of the victim (web server in our lab) are significantly depleted over the bandwidth of attacks received. A similar result was observed by performing *Single Request HTTP Flood* where multiple HTTP requests were made using a single session, masking them in a single packet. Programming languages offer multiple ways of creating HTTP requests. For example, the *Python* language offers the *requests*, *urllib* and *sockets* packages from which HTTP requests can be made. We experiment with all three variations of the packages and find approximately

the same results with the maximum number of requests. However, we preferred to use the *sockets* package as it offered multiple options for setting the payload, and the max number of requests was achieved through controlled threading.

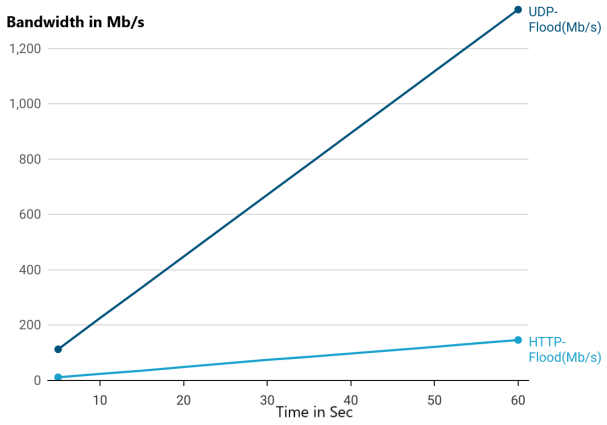

Figure 9. Estimated bandwidth comparison between HTTP and UDP flood

6.1.2. UDP-Flood. We try performing a controlled UDP-flood attack through the IDEs and find that some IDEs block UDP-based traffic. However, we were able to run UDP-based flood attacks from 18 IDEs in our experiment. We run the UDP-flood for a limited period of five seconds and observe the attack bandwidth ranging up to *1320 Mb/s* with the CPU load steadily increasing to an average of 22% per second. Figure 9 shows the estimated bandwidth of attacks received over time from HTTP and UDP requests. The estimation is based on the attacks received during our controlled experiment. We find that UDP-based attacks caused a higher impact on the victim's resources, leading to quicker service disruption than the HTTP requests. Note that the attack bandwidth could be higher as we used a controlled test script in our experiment, and the requests originated from a single online-IDE instance.

6.1.3. Multi-vector attacks. Multi-vector attacks involve using multiple flood-type attacks to achieve maximum bandwidth. While multi-vector attacks are ideal for achieving higher bandwidth by weaponizing, the supporting IDEs are a high-risk environment. We try the possibility of combining the HTTP and UDP-flood attacks from the IDEs. Our experiment shows that almost 55% of the IDEs are vulnerable to multi-vector attacks.

6.1.4. Text-encoding attacks. Boucher et al. [6] recently proposed a new type of attack in which the source code is maliciously encoded to appear different to a compiler than to a human. The authors present a proof of concept in multiple languages: Python, Java, JavaScript, Go, C#, C++, and Rust. We try the exploits with the IDEs to see if they support text-encoding standards like Unicode to manipulate the compiler-view. Our experiment revealed that 98% of the online-IDEs were vulnerable to the attack. We consider such vulnerabilities as potential discrete techniques for exploits. However, this is out of the scope of this work as we emphasize to availability rather than integrity attacks.

6.1.5. Other findings. We performed our experiment with the online-IDEs that support *Python* language scripting. Other than the unrestricted package imports, we also found unpatched versions of systems that are vulnerable to CVEs CVE-2020-14422, CVE-2020-8492, CVE-2019-9674, CVE-2013-1753 that can lead to Denial of Service attacks [25]. Furthermore, as some IDEs allowed the OS and system packages, we could obtain information about the host operating systems and the other packages. We further check for vulnerabilities using the version information we obtain and inform the owners about the vulnerabilities. Furthermore, we find user-authenticated IDE environments, which require user-signup, equally vulnerable as non-authenticated ones, though we exclusively consider the unauthenticated platforms in our experiment.

6.2. Comparison with amplification attacks

Amplification attacks involve an amplification factor that enhances the original attack vector to multiply the initial attack. While these attacks are known to amplify over UDP-based protocols like DNS, more recently, there is research where TCP-based protocols can be leveraged [26]. In this work, we use vulnerable IDE instances to contribute to an existing attack process by potentially weaponizing uncontrolled online-IDE environments. Although the IDEs can contribute more attack requests, it is not similar to an amplification factor. Our approach implies that the attacks can be magnified in numbers by spawning multiple IDE instances and threads on the go without compromising the system or injection of malware. However, similar to DoS attacks, the attacker's identity remains hidden as the source of the attacks traced back to the IDEs.

6.3. Implications

Our work suggests that uncontrolled online-IDEs can be leveraged as an open system for availability attacks by botnets. Through our observation, we find that many online-IDEs possess high, scalable resources that can be exploited for carrying out attacks on the Internet and for crypto mining purposes. Although the number of uncontrolled online-IDEs is not as significantly high as the number of vulnerable devices employed by massive botnets, we believe that uncontrolled online-IDEs can be used as a magnitude factor since they are an accessible resource. While there is no evidence of bots using such environments for attacks, we proactively identify the vulnerabilities and ethically disclose them to the owners to prevent such exploitation by adversaries. Furthermore, the primary reason for uncontrolled execution is a result of misconfigured environments. As online-IDEs are considered as a platform for learning, many security implications are overlooked in order to achieve similarity to that of local IDE environments.

Using uncontrolled IDEs provide attackers with discrete ways of launching availability attacks. In comparison to other vectors used for attacks, uncontrolled IDEs

provide the following advantages: *(i)* uncontrolled online-IDEs provide direct execution environments without any compromise steps to be undertaken by the attackers, *(ii)*, online-IDEs are equipped with reasonable resources that can facilitate attacks, *(iii)* multiple sessions of online-IDEs can be created to increase the magnitude of the attacks in case of ephemeral instances, *(iv)* uncontrolled online IDE environments are simpler to find on the Internet and do not require aggressive probing to find vulnerabilities, *(v)* since we observe no CAPTCHA checks in the IDEs on our findings, attackers can avoid any bypassing mechanisms that limit the botnets.

6.4. Limitations

Our approach utilizes online-IDEs on the Internet for availability attacks. As these IDEs are hosted on private infrastructure, evaluating availability attacks is ethically challenging. To address the ethical challenges, the evaluation of our approach involves some limitations. Firstly, we use a limited number of identified IDE instances to evaluate our methodology. This limits the full potential of the possible impact of the attacks. Second, we use rate-limiting in our test code to not cause any possible disruptions in the IDE service. This limits the use of the resources on the online-IDEs infrastructure. Third, we use an estimation-based approach to predict the possible number of attack requests per second achieved by multiple IDE instances running in parallel, which does not provide enough accuracy in the calculations and may have a high error rate. Lastly, we acknowledge packet drops occurring at intermediary devices in some of our experimental trials and discard environments that affect the overall throughput. Through this work, we intend to disclose the impact of running such environments to the owners and proactively prevent misuse of resources. It is a challenge to measure the impact of our approach in an ethical manner. We extrapolate the measurements received on a limited time-based experiment to accommodate the impacts to the IDE owners. We further acknowledge that many unknown factors may influence the values in our experiments. Our method is an honest attempt to identify uncontrolled IDE environments and prevent their misuse.

7. Ethical considerations & countermeasures

It is challenging to test our methodology as it involves sending high traffic from the Internet that may disrupt availability. We follow several precautions to avoid such a scenario. In this section, we discuss the ethical considerations followed in our approach.

7.1. Attack testing

We follow multiple steps in our attack testing approach to ensure that the availability of the online-IDEs or the intermediary networking systems are not compromised. The ethical measures we follow in our methodology are summarized below.

7.1.1. Informed consent. We take consent from the online-IDE owners to perform our experiment. We obtain consent from 18 IDE owners to perform our experiments. We assure the IDE owners of non-malicious experiments and measures to prevent resource exhaustion. Furthermore, we test a limited number of IDEs in our experiment, although we find many vulnerable uncontrolled online-IDEs.

7.1.2. Limited threads (rate-limiting). We run an HTTP-flood program to test the possibility of achieving maximum requests from the IDEs. However, we limit the number of threads in our program to prevent resource exhaustion. We use an estimation-based approach to ethically predict the number of requests that can be achieved from the IDEs.

7.1.3. Lab infrastructure for testing. We set up a web server in our lab infrastructure to target all the requests from the IDEs. However, we understand that this does not fully comply with the challenge of reducing the disruption in the network due to the traffic from the IDEs. We try to reduce the disruption by limiting the number of threads and the runtime of the experiment. The website used for measuring the HTTP flood requests received from the IDEs contained the necessary information about our experiment.

7.2. Responsible disclosure

We perform responsible disclosure to all the owners of identified uncontrolled online-IDE execution environments. The disclosure informs the owners of the importance, criteria, vulnerabilities, and proof of concept to test the environments independently. Furthermore, we perform an early disclosure to certain critical service providers (for example, a leading database service) that have a high possibility of traffic, even if this entailed the possibility of not using these environments for testing our approach. Additionally, we ask for IDE owners' consent to experiment on uncontrolled environments before they patch their systems. Until the time of submission of this paper, we hope that most of the uncontrolled IDE environments are patched.

7.3. Countermeasures

In this section we propose and discuss countermeasures against the criteria defined for uncontrolled execution environments.

7.3.1. Restricted file operations. File operations are essential for the import and export of data. However, it is crucial to restrict the operations and limit access to critical paths of the file system by simply employing containerized environments. Many online-IDEs use file operations for importing the source files; it is also essential to perform validation to scan for potential malware. An adversary can leverage unrestricted operations to either download malware or spread the malware to external systems.

7.3.2. Limited package support. Adversaries can use packages to perform malicious operations on the host machine, like downloading malware, accessing the host's file system, and scanning the network. Developers use

packages to support additional operations or import external libraries not part of the default package list. It is also crucial to limit the features of default libraries (for example, the sockets package) to restrict access to the network and limit the import of external libraries. While we acknowledge that limiting the functionality of libraries is a hard problem, we suggest to limit the attacks that leverage packages, by configuring network rate limiting in addition to memory and CPU resource limiting. Linux environments provide default tools for limiting the system resources per process. Administrators can further use containerization of individual user sessions to limit resource usage.

7.3.3. Bounded resource consumption. Limiting the resources per user and program is required. Unlimited resources can lead to disruptive operations on the host and be leveraged as an attack source. Also, limiting the number of threads that can be created ensures controlled resource usage. Furthermore, the use of timeouts that restrict the execution period ensures that a program does not run for extended periods and prevents flood-type attacks.

7.3.4. Sandboxed environments. Sandboxed environments ensure no access to external systems, and the user sessions are isolated from the other sessions running on the online-IDE. Each user has a dedicated isolated environment that is purged after the user session expires. Online-IDEs can leverage containerized environments to achieve sandboxing of individual user sessions and purge them after the end of the session.

7.3.5. Stateful user sessions. Online-IDEs run over a web service and can be configured to maintain stateful user sessions to track idle or disconnected users for stopping the program execution. This prevents bots from spawning multiple IDE instances to inject malicious code and exit the session to save resources on the bot client. Maintaining stateful user sessions can also help limit the number of sessions per user and limit resource usage.

7.3.6. Other measures. We accessed the IDEs through the Tor network and found that 98% of the online-IDEs identified allowed access. To limit suspicious events, we suggest the use of *CAPTCHAs* to verify the source of traffic and also limit the execution of suspicious code. We further suggest online-IDEs to integrate *CAPTCHAs* for validating each user session irrespective of the network to limit the bot activity. Moreover, we strongly recommend that all the online-IDEs have user authenticated sessions to prevent unnecessary resource usage. Lastly, to defend against text-encoding attacks, we recommend following the countermeasures suggested by Boucher et al. [6] to prohibit the support for text directionality control characters both in language specifications and in compilers implementing these languages.

8. Conclusion

This work identifies online-IDEs that offer uncontrolled execution environments that can be leveraged to perform availability attacks. We perform an Internet-wide search for online-IDEs and filter them by executing a test script that satisfies the uncontrolled execution criteria. Furthermore, we perform experiments to verify the possibility of availability attacks through the online-IDEs by informed consent. The estimated impact of the attacks is calculated by measuring the requests obtained from the experiments. We emphasize the consequences of having uncontrolled execution environments and proactively conduct experiments to assess the impact through this work. Lastly, we perform immediate ethical disclosure to the IDE owners to prevent misuse of the environment. As future work, we plan to generate scripts that can be used for checking uncontrolled execution of online-IDE environments such as permissions set for code execution, maximum file size that can be written, paths accessed, and max network bandwidth to help the administrators.

Acknowledgment

As part of the open-report model followed by the Workshop on Attackers & CyberCrime Operations (WACCO), all the reviews for this paper are publicly available at https://github.com/wacco-workshop/WACCO/tree/main/WACCO-2022

This research was supported as part of COM^3, an Interreg project supported by the North Sea Programme of the European Regional Development Fund of the European Union. We would like to thank Reinholdt W. Jorck og Hustrus Fond and Otto Mønsteds Fond for facilitating the research. Lastly we thank the reviewers for their valuable comments.

References

[1] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the mirai botnet," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1093–1110. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis

[2] G. Pellegrino, C. Rossow, F. J. Ryba, T. C. Schmidt, and M. Wählisch, "Cashing out the great cannon? on Browser-Based DDoS attacks and economics," in *9th USENIX Workshop on Offensive Technologies (WOOT 15)*. Washington, D.C.: USENIX Association, Aug. 2015. [Online]. Available: https://www.usenix.org/conference/woot15/workshop-program/presentation/pellegrino

[3] L. Ifigeneia, T. Marianthi, and M. Apostolos, "Enisa threat landscape 2021," *ENISA*, vol. ETL2021, pp. 46–50, 2021. [Online]. Available: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2021

[4] P. Wainwright and H. Kettani, "An analysis of botnet models," in *Proceedings of the 2019 3rd International Conference on Compute and Data Analysis*, 2019, pp. 116–121.

[5] R. Saini, S. Bali, and G. Mussbacher, "Towards web collaborative modelling for the user requirements notation using eclipse che and theia ide," in *2019 IEEE/ACM 11th International Workshop on Modelling in Software Engineering (MiSE)*, 2019, pp. 15–18.

[6] N. Boucher and R. Anderson, "Trojan Source: Invisible Vulnerabilities," *Preprint*, 2021. [Online]. Available: https://arxiv.org/abs/2111.00169

[7] Z. Alizadehsani, E. G. Gomez, H. Ghaemi, S. R. González, J. Jordan, A. Fernández, and B. Pérez-Lancho, "Modern integrated development environment (ides)," in *Sustainable Smart Cities and Territories*, J. M. Corchado and S. Trabelsi, Eds. Cham: Springer International Publishing, 2022, pp. 274–288.

[8] L. Wu, G. Liang, S. Kui, and Q. Wang, "Ceclipse: An online ide for programing in the cloud," in *2011 IEEE World Congress on Services*. IEEE, 2011, pp. 45–52.

[9] P. Chinprutthiwong, R. Vardhan, G. Yang, Y. Zhang, and G. Gu, "The service worker hiding in your browser: The next web attack target?" in *24th International Symposium on Research in Attacks, Intrusions and Defenses*, ser. RAID '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 312–323. [Online]. Available: https://doi.org/10.1145/3471621.3471845

[10] J. Liu, Z. Zhao, X. Cui, Z. Wang, and Q. Liu, "A novel approach for detecting browser-based silent miner," in *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*. IEEE, 2018, pp. 490–497.

[11] J. Rüth, T. Zimmermann, K. Wolsing, and O. Hohlfeld, "Digging into browser-based crypto mining," in *Proceedings of the Internet Measurement Conference 2018*, ser. IMC '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 70–76. [Online]. Available: https://doi.org/10.1145/3278532.3278539

[12] D. Georgoulias, J. M. Pedersen, M. Falch, and E. Vasilomanolakis, "A qualitative mapping of darkweb marketplaces," in *Symposium on Electronic Crime Research (eCrime)*. IEEE, 2021.

[13] L. Wu, G. Liang, and Q. Wang, "Program behavior analysis and control for online ide," in *2012 IEEE 36th Annual Computer Software and Applications Conference Workshops*, 2012, pp. 182–187.

[14] V. Kiriansky, D. Bruening, and S. Amarasinghe, "Secure execution via program shepherding," in *11th USENIX Security Symposium (USENIX Security 02)*. San Francisco, CA: USENIX Association, Aug. 2002. [Online]. Available: https://www.usenix.org/conference/11th-usenix-security-symposium/secure-execution-program-shepherding

[15] M. Pass. (2021) Icecoder. [Online]. Available: https://github.com/icecoder/ICEcoder

[16] Microsoft. (2022) Visual studio code. [Online]. Available: https://github.com/microsoft/vscode

[17] Coder. (2022) Code-server. [Online]. Available: https://github.com/coder/code-server

[18] L. Siira. (2022) Atheoside. [Online]. Available: https://github.com/Atheos/Atheos

[19] J. Zhang, J. Notani, and G. Gu, "Characterizing google hacking: A first large-scale quantitative study," in *International Conference on Security and Privacy in Communication Networks*, J. Tian, J. Jing, and M. Srivatsa, Eds. Cham: Springer International Publishing, 2015, pp. 602–622.

[20] Censys. (2021) Censys search. [Online]. Available: https://censys.io/

[21] SHODAN, "Shodan," 2021. [Online]. Available: https://www.shodan.io/

[22] A. B.V. (2022) Ace editor. [Online]. Available: https://github.com/ajaxorg/ace

[23] J. F. Balarezo, S. Wang, K. G. Chavez, A. Al-Hourani, and S. Kandeepan, "A survey on dos/ddos attacks mathematical modelling for traditional, sdn and virtual networks," *Engineering Science and Technology, an International Journal*, vol. 31, p. 101065, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2215098621001944

[24] CloudFlare, "Meris botnet," 2021. [Online]. Available: https://blog.cloudflare.com/meris-botnet/

[25] C. Details, "Cve details," 2021. [Online]. Available: https://www.cvedetails.com/vulnerability-list/vendor_id-10210/product_id-18230/year-2020/opdos-1/Python-Python.html

[26] M. Kührer, T. Hupperich, C. Rossow, and T. Holz, "Hell of a handshake: Abusing TCP for reflective amplification DDoS attacks," in *8th USENIX Workshop on Offensive Technologies (WOOT 14)*. San Diego, CA: USENIX Association, Aug. 2014. [Online]. Available: https://www.usenix.org/conference/woot14/workshop-program/presentation/kuhrer