Article

Use and Abuse of Personal Information, Part I: Design of a Scalable OSINT Collection Engine

Elliott Rheault, Mary Nerayo, Jaden Leonard, Jack Kolenbrander, Christopher Henshaw, Madison Boswell and Alan J. Michaels *

Virginia Tech National Security Institute, Blacksburg, VA 24060, USA
* Correspondence: ajm@vt.edu

Abstract: In most open-source intelligence (OSINT) research efforts, the collection of information is performed in an entirely passive manner as an observer to third-party communication streams. This paper describes ongoing work that seeks to insert itself into that communication loop, fusing openly available data with requested content that is representative of what is sent to second parties. The mechanism for performing this is based on the sharing of falsified personal information through one-time online transactions that facilitate signup for newsletters, establish online accounts, or otherwise interact with resources on the Internet. The work has resulted in the real-time Use and Abuse of Personal Information OSINT collection engine that can ingest email, SMS text, and voicemail content at an enterprise scale. Foundations of this OSINT collection infrastructure are also laid to incorporate an artificial intelligence (AI)-driven interaction engine that shifts collection from a passive process to one that can effectively engage with different classes of content for improved real-world privacy experimentation and quantitative social science research.

Keywords: open-source intelligence; OSINT; privacy experimentation; personal information

1. Introduction

Online users consistently share personal information (PI) with Internet entities, despite hoping for security and anonymity. From online social networks to news and entertainment subscriptions, the vast majority of Americans are faced with the inherent trade of their PI for the resulting benefits in agreeing to customized online services (while not all personal information (PI) is personally identifiable information (PII), the aggregation of the more broadly defined PI through online transactions is foundational to privacy or tracking operations). It is widely recognized that users’ PI, as well as their online behaviors, are being tracked while web browsing. Previous studies identified more than 500 different tracking methods used by different sites, and certain pages have trackers that are connected to multiple parties [1]. Beyond this consensual sharing of our PI [2], we inherently run the risk of data breaches [3,4], insider threats [5], corporate mergers or bankruptcies [6], or good old-fashioned misuse [7] leading to the unanticipated release of PI. It is estimated that the average person has hundreds of possible threat vectors for the release of their PI used in establishing those accounts. Given the difficulty of identifying who is responsible for sharing our personal information when we receive spam or scams in such an environment, how do we disentangle the chaos to identify the bad actors?

The broader class of OSINT research, sometimes characterized as a class of big data research [8], seeks to derive actionable intelligence from the correlations in large bodies of data, often fusing results to help answer a specific question. Most often, this process is passive, suggesting that there is “no engagement with the target, passive collection from publicly available information, and low risk of attribution” [9]. However, the extension to active OSINT is well recognized as a potential way to derive additional intelligence, yet also comes with risks to bias the result or other ethical considerations; the vast majority of published OSINT research is based upon passive collection.
For the specific category of research in personal privacy, using personal information of real users would be a significant concern given the potential harms that come from its release. In other cases, the goal of OSINT research has been to develop predictive analytics from correlations to open-source content, such as that performed by the IARPA EMBERS program [10] for the 2013–2014 uprisings in Latin America [11]. The importance developing quantitative frameworks for OSINT results is also recognized, with attempts to derive answers for operational situational awareness [12], for business risk assessments [13], or cybercrime [14]. In other cases, dedicated tools are developed for better visualization of the results [15] so that we can more effectively ingest the results.

The management of the underlying data in OSINT research is one of the most crucial components, with poorly labeled datasets significantly impairing the potential results. Even the management of such data warrants thorough publication [16]. Similar challenges exist between OSINT research and artificial intelligence/machine learning research, where even small amounts of poisoned data can reduce the effectiveness of the algorithms [17]. As a result, there is a strong desire in any large-scale cybersecurity [18] or quantitative social science OSINT effort to ensure that data labels are well understood and sufficient for the planned data analysis. Numerous commercial tools exist for exactly this purpose, though most are tailored to small classes of applications. Once a sufficient collection of well-labeled data can be obtained, the final challenge of OSINT research is marrying it to real-time computing platforms. The enterprise scale at which collection, processing, labeling, and curation must proceed to enable near real-time decisions demands much more than traditional desktop computing. This paper is similar to other frameworks [19] that seek to accelerate OSINT research with dedicated collection and processing infrastructures.

1.1. Ethical Considerations

Before proceeding with the framework design, studies have been performed to help navigate the ethical considerations when applying fake personas for research purposes. Most of this work aims to protect stakeholders in online social networks. In such cases, the ethical considerations deal with protecting users, the providers of the service, and the advertisers/investors in such services. In terms of research, there are ethical ramifications regarding other users and their consent to be active in the study being conducted, and the indirect exposure of their information. Outside the exposure of information, there is also the wasting of time or resources of other users or third parties invested in the network. An army of fake bots would actively be violating the agreements required for the service and waste server resources in satisfying the requests of fake users. Further, depending on the influx of fake accounts, statistics that drive value and advertiser spending could be impacted [20]. Researchers in several cases have managed to infiltrate private organizations through strategic social bots targeting users involved in the organization [21].

Critics of this kind of information gathering have noted that a more ethical route would be designing a closed model to determine how vulnerable information might be taken or shared, but previous research has shown that utilizing fake accounts in the real world is far more effective and can be performed if certain limitations in size and scope are applied [22,23]. Using fake identities is a potent avenue of information gathering and enters into a challenging ethical arena when for most use cases of the technology, effective data collection requires some level of deception. However, various international review boards have begun classifying risk levels to the implementation of fake identities for studies, especially in regards to OSNs. The three categories of research in this area are observational, interactive, and survey/interview [24]. Our use case falls under the observational research category, in that identities are being utilized as an instrument to safely observe how different entities interact with users and leverage their PI. Another framework is the Department of Homeland Security Ethics and OSINT Scorecard that attempts to quantify adherence with guiding ethics principles [25]; our self-evaluation of the 20 criteria yielded a score of 86 on scale of 20–100, suggesting the U&A effort “likely excels at adhering to ethical OSINT policies”. Using real information to conduct such a study would be unethical according to
several ethical guidelines that warn against placing compromising information at risk. We have, therefore, bounded our fake IDs by intentionally limiting the scope of information used in creating them (e.g., no social security numbers and drivers license numbers), and actively de-validating any data that could be traced back to a real source, such as our random address generation. In addition, though the project aims to generate large numbers of IDs, only a small group of IDs are dedicated to fake accounts at any one service provider, minimizing impact to the hosts. Moreover, the types of transactions are primarily machine to machine, rather than targeted at humans. Finally, we have unofficially reviewed the approach with our university Institutional Review Board (IRB), particularly when dealing with dark web or questionable content, and received feedback that no protocol was necessary given the fake PI. Should other experiments or operations legally require fake identities that possess such information, the assignment of the additional information can follow similar models to those described previously.

1.2. Paper Outline

Given this context, the Use and Abuse of Personal Information (U&A) framework was constructed to expand the quantitative support for privacy experimentation and social science research. In our work, we have found the need for an expanded data collection framework that can be an active participant in the content dissemination rather than a passive observer. This paper describes the design, implementation, and early validations of the U&A framework via a wide range of quantitative social science experiments.

The overall system architecture for the U&A framework is introduced in Section 2, with emphasis on the design requirements for scalability, capabilities, and limitations. Detailed descriptions of key components, including the real-time signup engine, the collection servers supporting email/voice/text modalities, and the curation of received content into labeled datasets are then presented in Section 3. A deep dive into the human-in-the-loop account signup engine is explored in Section 4, emphasizing how the experimental design, fake IDs, and collection system are fused to perform active OSINT operations. The overall conclusions and future work, which emphasizes the nearing-completion account interaction engine, are provided in Section 5.

2. U&A System Architecture Overview

The core elements of the U&A infrastructure, shown in Figure 1, are the generation of fake IDs, the assignment of those fake IDs via the signup engine, the email/voicemail/SMS collection servers, and the realistic fake ID interaction via the account interaction engine. Significant emphasis is given to the storage and normalization of the raw data received from the collection servers to produce a secure labeled database, while the general purpose application programming interface (API) defines the interactions with said database. This allows our system to be abstracted across many research questions, via the generic API, ensuring minimal development effort between questions. Core requirement assumptions within this infrastructure include:

- Fake identities are generally limited to a single one-time online action (i.e., transaction), where they provide some element of their PI to help ensure that we do not provide an undue burden to any external organization. The only deviation from this receive-only posture is the account interaction engine, which enables the transmission of tracking cookies/clicking links embedded in the received content for a limited number of accounts. Future interactions may also include AI-crafted responses for content.

- Individual fake identities and all of their associated data are maintained consistently through the infrastructure via unique identifiers (UIDs). If relevant to specific research questions, the characteristics of pseudorandomly generated fake identities may be sculpted (e.g., over-writing all addresses to be located within a chosen congressional district) to test the desired hypotheses.
The overall Use and Abuse (U&A) architecture was developed with the plan to perform enterprise-scale active OSINT collection for email, phone/voicemail, and SMS text content, which can be fused with other data feeds such as web scraping, social media feeds, or similar passive content. This paper focuses on that active OSINT collection, which includes the generation of fake identities, assigning those identities uniquely to research questions, accelerating the signup process with specialized human-in-the-loop tools, and then the subsequent collection (yellow box of Figure 1) of the data content received via email, phone/voicemail, or SMS text.

Over the years, we have refined the previous fake ID models by using national demographic distributions, U.S. Census-derived names and addresses, and associated unique personal characteristics that might be requested online. We can assign these identities for use in multi-disciplinary experiments that gauge how fake PI propagates across the Internet, testing questions of the conditional treatment of accounts possessing (a) different demographic information; (b) account access behaviors and/or activity levels; (c) geographic locations (e.g., GDPR and California-based privacy protections); (d) different account types, industries/business models, or third-party size; and (e) minimal and maximum sharing policies. Additional details of the fake ID generation process (green box in Figure 1) and considerations are captured in a companion paper [26].

Another key part of the U&A framework is an account signup engine (blue box in Figure 1) that enables accelerated online transactions. Trading the complexities of performing online transactions (signups, purchases, etc.), we chose for this to be a human-in-the-loop process, where the fake ID content is organized for rapid ingestion at scale. The current signup engine supports real-time interaction with hundreds of students performing signups, including access to two-factor verifications received by text, phone, or email.

The experiment design is exceedingly open ended, with large groups of small teams, typically undergraduate teams led by a faculty member, all focused on identifying which online interactions are needed to answer a specific research question. This translation of
a specific social science topic or questions can be broken down into (a) concrete research hypotheses, (b) data collection plans (interactions with organizations through the U&A platform and other OSINT sources), (c) multi-disciplinary data analysis objectives, and (d) culmination of the research into a peer-reviewed paper submission. Throughout this process, student teams receive expert guidance to help them refine the hypotheses and ensure balance in the data collection to minimize inherent biases. A common research design can be followed as depicted in Figure 2, which shows a generalized research flow (top), mirrored by a specific recent experiment that combined political science questions with machine learning (ML)-based sentiment analysis tools to quantify how employee donations correlate with corporate messaging (bottom).

![Figure 2. Research flow (top) and specific example (bottom) of multi-disciplinary research topics with faculty/student engagement.](image)

Beyond this passive collection process, part of the ongoing research is the expansion of a real-time interactive response engine that nearly mimics the behaviors of a human recipient. With such capabilities, the AI-driven computational power of the platform, coupled with the creativity of students from across all majors, can lead to a multi-disciplinary computing platform well suited to a wide range of students.

3. U&A Framework

At the heart of the U&A engine is the enterprise-grade collection services, consisting of a custom-built enterprise-scale email server (unique accounts for each fake identity) and a cluster of three FreePBX phone servers (Clearly IP 790) supporting 6000 voice-over-IP (VoIP) phone lines with inbound SMS messaging. By maintaining unique email accounts for each fake ID, we are able to attribute the incoming content back to the specific one-time transaction performed by each email. However, supporting phone lines are more difficult. This is due to their complex configuration and storage, rigid architecture, and monthly expenses due to the required third party service (ClearlyIP). Additionally, due to branded campaign laws, the recurring cost of these phone numbers is exponentially higher than just acquiring the numbers. We will discuss this more in depth in a later section.
Following the collection processes, a series of preprocessing services are run across all raw data to help formulate an organized labeled database which can be easily ingested by non technical researchers. Emails are batch processed to convert the raw HTML text into a plaintext format, parse out all hyperlinks, and serializes the attachments (including tracking pixels) to be pulled for subsequent analysis. The SMS data are pulled from ClearlyIP’s internal MariaDB instance over the network via a MySQL connector and stored in the post-processor’s Mongo database. The voicemail data are then pulled in using an export routine in conjunction with scp (secure copy protocol). Once pulled over, the voicemail data are passed through a variety of audio processing tools to transcribe the data into text, which is then saved in the post-processors database. In addition, all of the raw content is stored in case there is a need for subsequent analysis later. Both the raw and processed data are mounted on a raided network attached storage device (NAS) to ensure data integrity through corruption and hard drive failure.

Content analysis of the results is highly dependent on the research question, typically including statistical analysis of which fields of the fake IDs are transmitted, as well as manual analysis of the preprocessed data from the collection engine; the flexibility of the collection processing thus allows for virtually any research topic that seeks information collected from online interactions to be received in a common U&A infrastructure. Finally, the cybersecurity-focused analysis of content is performed on the interaction engine within a Proxmox-hosted VM infrastructure to identify any anomalous or potentially malicious content (virus scanning and/or custom tools), and/or perform limited interactions with the sender. We will limit these interactions primarily to point-to-point responses where the received content has solicited our response (e.g., hyperlinks and tracking pixels). Such techniques can be viewed as account keep-alive interactions and for the improved understanding of the linked content delivered to us.

While our initial pilot experiment [27] featured only 300 fake IDs, the current architecture targets the creation and distribution of millions of fake IDs, supporting experiments in virtually every corner of the internet. To facilitate an experiment at this scale, both the email server and the phone server must be capable of handling many thousand individual accounts all receiving messages simultaneously. Furthermore, any service must be hosted locally to preserve the integrity of our fake IDs within our experimental paradigm of “one fake ID transmitted in a single online transaction”.

Email and phone solutions of this scope rival university-wide, enterprise-grade solutions with dedicated support teams comprising several technicians and managers. Our system, in contrast, must operate with absolutely minimal oversight. In addition, the solution must be reasonably hardened against malicious actors, as our experimental approach runs the risk of drawing the ire of spammers and hackers. Furthermore this enterprise-grade system must be lightweight, able to run on a single entry-level ($5K) server. Lastly, the software solution must be easily scalable; the rapid cadence of our experimental question generation by student teams requires the generation of thousands of new email accounts and phone numbers on a short time scale. All these considerations require a careful balancing act that results in the solution described in Sections 3.1 and 3.2.

3.1. An Enterprise-Scale Email Server

The receiving and processing of emails is one of the largest components of the U&A framework. With the expansion from the 300 fake IDs used in the pilot study [27] to 100,000 fake IDs, the email server needs to be re-implemented in a highly scalable fashion. This section outlines our updated requirements, the limitations of the prior approach, and the research and development of the new solution.

3.1.1. Requirements and Limitations

To support our new baseline of 100,000 fake IDs, the email server must be (1) highly scalable, (2) require 0 human interaction for collection, (3) easily support the addition and deletion of fake IDs en masse, (4) completely self-hosted, (5) highly configurable, and
run on our proxmox environment. The previous solution [28] utilized a Rainloop-based mail instance, which compiles emails and voicemail messages into a single account. With the 16,540 total collected emails being saved to a single account in addition to the limited filtering provided by Rainloop, a large amount of manual interaction is required to associate the emails with their corresponding fake IDs. The single inbox also prevents the ability to send email from specific fake IDs. Lastly, the rigidity of the email managing sieve scripts coupled with the lack of storage and backup features leads us to find an alternative.

3.1.2. Infrastructure Research

Rainloop only provides us with an email client to access our emails with; we need to host everything for our new design. Self-hosting gives us full control over our design and enabled us to scale and deploy without external restrictions. In order to self-host, we have to utilize our own mail transfer agent (MTA) and mail delivery agent (MDA). These two programs facilitate the transfer and delivery of emails onto our own system, where we can access them in any way needed. This additionally helps address the constraints of scalability, configuration, and automation of our systems. The MTA needs the ability to control who can send messages. This requirement would help ensure that collection-only research questions are not sent in emails. The MDA has the requirements of reliability, flexibility, IMAP protocol access, MIME attachment support, and physical file access. Reliability and flexibility ensure our MDA is configurable without issues like data loss. IMAP enables users to view emails remotely if needed for a research question. MIME attachment support allows us to collect more attachment data types. Physical file access helps facilitate backups and automation. Additionally, both software need to be open source so we can use and modify the software without license overhead.

After extensive research, we procure five software options: Dovecot, Mailu, Mailcow, Modoboa and iRedMail. Mailcow and Mailu are docker-based groupware containing all the functionality we need; Modoboa is a Python-based manager, and iRedMail is a full-server deployment. We choose Dovecot and Postfix for our MDA and MTA. The reason for choosing Dovecot and Postfix only versus an entire suite is simplicity. The other software suites utilize both programs for their MTA and MDA. With this in mind, we find vast documentation for the individual programs. Many of the features the other suites list are either possible with Dovecot or unnecessary. For example, antivirus and spam filtering are not a part of our experimental design since we do not open emails on the server. Our original design does not require us to manually open emails, so IMAP protocol could be used instead of self-hosting a web client. Additional research reveals the Docker Mail Server (DMS) to us, which is another docker-based groupware that meets all our needs.

DMS is a docker-based, full-stack mail server that deploys with configuration files only. DMS combines all configurations in Dovecot and Postfix into a single environment file, allowing us to reference and change the configurations quickly. With Docker-Compose, we can set up and tear down the container quickly and utilize volumes. Volumes provide easy access to the emails created, enabling us to automate preprocessing when needed. While DMS is also docker-based groupware like Mailcow and Mailu, a crucial difference between them is that DMS strictly reconfigures files. This makes it easier to configure and set up the mail server. Another difference is the number of components used. DMS contains fewer additional services compared to the other two. These components could also be disabled in configuration, enabling us not to include services like antivirus or spam filtering without a workaround. Finally, DMS has verbose documentation and simple command-line interactions. The biggest concern from the change from standalone daemons to DMS is the inability to store emails in a Mbox format. The previous experiment stores emails in that format. While this change does not change the contents of an email, the preprocessing subteam has to develop a different way to parse emails.

Since we are self-hosting our software, we need our own hardware as well. Email servers are not resource intensive. This enables us to partition our mail server on a Proxmox environment with minimal resource requirements. Using the previous project as a baseline,
we estimate that the worst-case scenario of emails received concurrently would be 11.6 per second for 1 M accounts, assuming each account receives one email daily. Storage would also be a considerable constraint, with around 27.5 TB of data a year estimated to be collected in a worse-case scenario.

In order to address our storage needs, we utilize network attached storage (NAS). NAS allows us to store large quantities of data in a RAID-based format, providing a scalable and safe way to store our data. Compression is handled efficiently if the CPU used has multiple cores and threads. RAM is not utilized in sending messages and was not considered a constraint at all. Initially, we consider a custom Supermicro server but find that a simple Dell PowerEdge R350 meets all of our constraints within a budget of $2K.

3.1.3. Proxmox Virtual Environment

The project is more than just an email server. Analysis, file management, and other services are needed to run the project. We implement our email server within a Proxmox Virtual Environment (PVE). Utilizing Linux containers (LXCs) within the PVE instead of separate containers creates a configurable partition between our services. Each LXC has its own connection to the network and OS configuration, allowing us to be specific about each container. Most of the containers are unprivileged, granting access to specific directories only. Every LXC, except the email server, is on a private network. This exposes our email services to the public only, protecting everything else on the same device. This allows us to use the rest of the resources on our system for other parts of the project. An example of this is the Nexus LXC. The Nexus LXC is responsible for backing up emails to a NAS and transferring them to a folder only the processing LXCs can see. An hourly cron job backs up emails to NAS on the private network and moves emails to a folder that the preprocessing containers can access. This increased functionality increases the complexity of our email server. Running a Docker container inside a Proxmox LXC is not recommended and is disabled by default. A solution to this is running the container in a VM instead. LXCs do not have access to VM data, making this impossible to utilize for our current design. Our solution is increasing the Docker PID number limit, adding the Docker daemon location to a LXC visible directory, and configuring the LXC to allow nesting.

3.1.4. Email Generation

One million identities is a large volume of email accounts. There is no possible way a single individual can populate these accounts quickly; this requires automation. Thankfully, DMS contains commands to create accounts. We create a Python program to solve this problem as represented in Figure 3. A CSV file containing the columns emailUsername and emailPassword can add or remove email accounts in bulk. The program works by iterating through the CSV file and calling the respective DMS command via the Python library subprocess. This process is indeterminate when scaled and struggles to perform when creating thousands of emails. After many email addresses are added, there is an issue with signing into accounts via IMAP and receiving email inboxes. Fixing these issues requires some change. The program is modified to call the Docker commands directly. The CSV comprises 1 M identities, split it into 1000 identity CSVs as well. This change enables us to stop and resume easily if an issue arises. The combination of these changes fixes the issue of accounts not propagating correctly.
3.1.5. Auxiliary Configurations

Basic security measures are taken to ensure that the rest of our systems will be safe if the email server is compromised. The email server has no access to the private network. By default, all undefined port connections are dropped. SSH capabilities are limited to public keys and can only be accessed within our VPN and academic subnets. The most likely point of compromise would be a bad actor within the research team that has credentials to the phone server. During signups, email confirmations populate on the server. Student participants in the signup event do not have access to the email server. A sieve script is used to redirect emails to a webhook, which allows a large number of individuals to sign up without directly accessing the email server.

3.2. A Custom VoIP Phone Exchange

The VoIP phone server plays the second primary role for our data collection methods for the experiment. This subsystem enables us to not only receive phone calls but also to proficiently record voicemails and capture SMS messages. Additionally, the significance of the phone server extends beyond data collection. A secondary, yet vital function is to enhance the success rate of fake identity signups by facilitating two-factor authentication (2FA) processes. We allocate unique phone numbers to distinct identities, thereby enabling us to obtain authentication codes and links. These authentication elements are pivotal in meeting the security requirements imposed by some online entities to which these identities are linked. This, in turn, broadens the project’s scope, granting us access to research online platforms that enforce more robust authentication mechanisms.

3.2.1. Requirements/Assumptions

Our first decision is selecting the type of connectivity to host our phone server. A locally hosted VoIP (Voice over Internet Protocol) server is deemed preferred over cellular communication for many reasons. First, it is more cost effective, meaning the possibility of scaling up the quantity of phone numbers we need for the experiment does not prevent execution on financial grounds. Next, VoIP has a flexible network infrastructure that allows users to access the services from different platforms; these include dedicated VoIP phones and several other devices. Its flexibility helps us customize elements of 6000 phone numbers uniformly and efficiently. Without this malleability, we would encounter an error-prone and time-consuming process to adjust the settings of the phone numbers and phone-answering profiles.

In terms of hardware, an on-site bare metal FreePBX Server is selected. Being an open-source Linux-based platform, we know that most devices (including Raspberry Pis) could test FreePBX configurations. The benefits are that we maintain the hardware on-premises and have full control over configurations and usage. Due to the unique nature of the experiments, we find it important to have control over the data to ensure no additional
sharing by a third-party service, such as the previously used Zadarma tools. On the other hand, we need to invest in the security, physical space, and supporting infrastructure that it takes to manage the server on site. The server (CIP 790 PBX) is selected for its large user capacity—2000 extension numbers. The CIP 790 PBX comes preloaded with the FreePBX Distro and has a call capacity of 350 concurrent calls. The hardware specifications include a Quad Core i7 Processor, 32 GB RAM, dual 500 GB SSD storage, multiple Ethernet ports (allowing flexibility for separated networks), all within a 1 U chassis design.

As a service provider, we compare and analyze six different SIP (Session Initiation Protocol) providers that would enable us to use SIP trunking. SIP trunking is the technology used in telecommunications to enable the delivery of VoIP traffic. Some positives of using SIP trunking are the powerful caller ID options and the quick development. ClearlyIP is chosen as the best aligned service provider option based on scalability and the payer-per-channel policy. During the design and planning phase of the experiment, we plan to dynamically scale the number of phone numbers based on new research questions and monitoring of activity to the phone lines; for lines where activity goes dormant for an extended period of time, we can release our lease on the number.

3.2.2. Server Setup and Management

Setting up a VoIP server requires several considerations to ensure reliable performance and security. The server comes with an intuitive GUI, which allows server users to avoid extensive Linux commands. Setting up the firewall is simplified, with verification performed by local experts.

The data collected by the VoIP server are key for experimental success, and therefore a heavy emphasis is placed on data management. There are four types of data collected: SMS messages, MMS messages, voicemail recordings, and call recordings. All SMS and MMS messages are initially stored in a MongoDB database containing a timestamp, recipient ID, sender ID, and message content field. For voicemail messages and call recordings, an additional log text file is produced and is stored with the recording in a directory based on date. These data are regularly backed up to our NAS and are transferred to our preprocessing system nightly for parsing.

3.2.3. VoIP Capabilities and Limitations

The VoIP capabilities and limitations can be broken down to three categories: communication, scalability, and reliability. For communication, we can send or receive voice calls, SMS/MMS, and voicemails. The only form of communication the server cannot handle is video calls, but for our purpose, this is not necessary. Additionally, when recording voicemails, the server allows custom waveforms to be played during the phone call. Using this, we create databases of prerecorded material and attempt to string along a given spam caller once we identify spam activity. This idea stems from Lenny, who is a bot designed to target telemarketers, scammers, and other unwanted incoming calls. The ability to scale the service up to 2000 individual phone extensions on one server allows collection across a large number of simultaneous research questions using different numbers. As our research has expanded, we have since added two identically configured servers to support up to 6000 different phone extensions.

3.2.4. Setting Up Phone Lines

VoIP direct inward dialing (DID) extension setup is simple because of the dynamic relationship between the server and ClearlyIP website, where DIDs are functionally added. After requesting a certain number of phone lines, the numbers are directly added to our account and can be added to the server at any time with a couple clicks and a refresh of the server. Reserving DIDs equates to the purchase of additional phone lines and requires a few (human-in-the-loop) clicks each.
3.2.5. SMS Limitations

Unlike emails, which can theoretically scale to any number, each phone number or DID has a $1 activation fee in addition to a $1 recurring monthly cost. In order to support SMS messaging, a DID must belong to a branded campaign, which has a $15 activation fee and a $3–12 recurring monthly cost and can hold up to 49 DIDs. However, through working with ClearlyIP, they allow us to forgo the branded campaign in cases where we only want to receive inbound SMS messages. This creates an additional consideration in the research question formation phase, as we have to weigh the need/impact of including phone numbers/outbound SMS messaging due to the additional costs they incur. Our past experiments have shown that a majority of questions require phone numbers, but not every fake ID in a question requires a phone number. This leads us to implement an API which manages a pool of preregistered phone numbers which can be dynamically assigned to fake IDs on signup. Outbound SMS messaging has yet to be needed in the OSINT collection engine but is likely to play a key role in the interaction engine.

3.2.6. Creating Phone Answering Profiles

In supporting a variety of research questions, we want to be able to reflect the personalities of our fake IDs, so we assign each a phone-answering profile. To create these, we incorporate variations in voice gender, number of rings, voicemail settings, and different recorded voices upon answering; these differences in behavior are in and of themselves an intriguing experiment that can shed light on scammers’ reactions and could make for valuable data by recording their changes in behaviors and the changes in the content being delivered over the phone. Varying the number of rings and enabling or disabling voicemail can also create uncertainty for scammers, allowing us to better understand how spammers behave. Similarly, we create profiles for the handling of SMS messages. Creating user profiles that adapt based on actions like opening or not opening messages, responding or not responding, forwarding SMS, opening SMS, and clicking on links is a strategic approach to personalizing one’s digital interactions as well as test differential sharing behaviors based on account activity. By automating responses or triggering specific actions based on these interactions, high quality data can be collected on how the spam we continue to receive changes. Furthermore, for both voice and text spam, several forms of analysis will be performed—this includes sentiment analysis and exploring the types of links (malicious or not) we receive, aside from determining who shares our PI.

4. Signing Up Fake IDs

Signup events, where fake identities PI is transmitted en masse to the desired recipient(s), are one of the most time-intensive components of the experiments if performed solely by humans. Portions of this process are intentionally difficult to automate, due to tools like CAPTCHA bot detection, 2FA, and the generalized application of the system. Hence, a semi-automated signup procedure is developed to accelerate and organize our signups. This signup engine handles the storage/delivery of fake identities, the state of the signups, and the retrieval of 2FA information, while its users copy the provided fake identities’ information to the login-forms, solve the CAPTCHA tests, and resolve the 2FA challenges with the information provided by the engine.

Signup events are scheduled so all users interact with the tool at the same time. This is for the purpose of minimizing any bias introduced by the differing times signing up. In addition, we restrict users to be on Virginia Tech’s network. This ensures that no malicious external connections occur, keeping the fake identities’ information secure. Once connected, users are greeted with the home page as depicted in Figure 4. Users are required to submit their first name, last name, and enter the event specific password. Once logged in, they are navigated to the blank signup page (Figure 5), where they are able to interact with the fake identities. This page consists of a table containing 29 of the default attributes held by all fake identities. The table also dynamically loads in any additional attributes, which are specially defined on a per-research question basis. Users then click the “Get
Fake Identity” button, sending an atomic request to the fake identity database, returning an unused fake identity (Figure 6). Next, users click on the table’s rows to copy a field’s value to their clipboard in addition to flagging it as used. That information is then pasted into the targeted website’s signup page. Upon completion, the users click the “Go to Survey” button, which navigates them to the survey page as shown in Figure 9. This page allows users to flag a signup as unsuccessful, defined by the inability to transmit any data which enable our collection engine to receive email or phone interactions. The user is also responsible for recording which phone number is submitted with the fake identity as well as any modifications performed to the fake identity during the signup. For example, if the provided password is “example_password” and the website does not accept “_” characters, then the user would change it to "example-password" and enter “changed password to: example-password” in the Other Notes section. Finally, the user clicks the “Submit Survey” button, saving all modifications to the fake identity database and navigating the user back to the blank signup page (Figure 5).

In cases where 2FA is required, users must complete them before submitting the survey. This process requires the users to request a two-factor code or link to be sent to the email address or phone number of the fake identity. Once complete, they can navigate to the “Two-Factors” page as shown in Figure 7. This page features a table containing every email which has been sent to a fake identities’ email address that the current signed in user has successfully requested. Users can sort the results by subject, sender, or date received, to help them find the email of concern. Clicking on a row will reveal the raw email file as shown in Figure 8, where users can locate any code/link necessary to complete the two-factor challenge. Finally, once a user is complete with the signup, they perform a survey as highlighted in Figure 9 to capture any irregularities or notes for later analysis.

Figure 4. Signup engine login page.
Figure 5. The signup engine’s blank signup page.
### Use & Abuse Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID Number</td>
<td>3006</td>
<td></td>
</tr>
<tr>
<td>URL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phone Number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>First Name</td>
<td>Bella</td>
<td></td>
</tr>
<tr>
<td>Last Name</td>
<td>Tessier</td>
<td></td>
</tr>
<tr>
<td>Age</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>Birth Date</td>
<td>Thu Jun 05 1969 00:00:00 GMT+0000 (Coordinated Universal Time)</td>
<td></td>
</tr>
<tr>
<td>Email Username</td>
<td><a href="mailto:bella_tessier803@uapli.com">bella_tessier803@uapli.com</a></td>
<td></td>
</tr>
<tr>
<td>Street Address</td>
<td>8274 W Rastin Rd</td>
<td></td>
</tr>
<tr>
<td>Town/City</td>
<td>Wittmann</td>
<td></td>
</tr>
<tr>
<td>State/Province</td>
<td>Arizona</td>
<td></td>
</tr>
<tr>
<td>Zip Code</td>
<td>85361</td>
<td></td>
</tr>
<tr>
<td>Country</td>
<td>USA</td>
<td></td>
</tr>
<tr>
<td>Website Password</td>
<td>2023jimVT2023I</td>
<td></td>
</tr>
<tr>
<td>Security Question 1</td>
<td>The Green Mile</td>
<td></td>
</tr>
<tr>
<td>Security Question 2</td>
<td>Blacksburg</td>
<td></td>
</tr>
<tr>
<td>Security Question 3</td>
<td>Floral White</td>
<td></td>
</tr>
<tr>
<td>Height</td>
<td>5'6&quot;</td>
<td></td>
</tr>
<tr>
<td>Weight</td>
<td>177</td>
<td></td>
</tr>
<tr>
<td>Sex</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>Sexuality</td>
<td>Bisexual</td>
<td></td>
</tr>
<tr>
<td>Pronouns</td>
<td>She/Her</td>
<td></td>
</tr>
<tr>
<td>Race</td>
<td>Black</td>
<td></td>
</tr>
<tr>
<td>Nationality</td>
<td>Non-Hispanic</td>
<td></td>
</tr>
<tr>
<td>Political Affiliation</td>
<td>Republican</td>
<td></td>
</tr>
<tr>
<td>Income</td>
<td>91109</td>
<td></td>
</tr>
<tr>
<td>Job Title</td>
<td>Industrial Machinery Repair Worker</td>
<td></td>
</tr>
<tr>
<td>Education Level</td>
<td>Masters degree</td>
<td></td>
</tr>
<tr>
<td>Education Major</td>
<td>Social Sciences</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 6.** Signup engine's populated signup page.
Figure 7. Table of two-factor authentication emails in the signup engine.

Figure 8. View of an email from the signup engine two-factor page.

Figure 9. Signup engine survey page.
4.1. Developing a Signup Engine

4.1.1. Design Choices

To achieve high portability and scalability, we decide to use a containerized architecture for the web application. Docker is chosen as our container engine due to its easy configuration, robustness, speed, and build/deployment management tools [29]. This approach allows us to build/deploy our software across any operating system or machine that supports docker engine. This effectively reduces the previous student prototype’s long list of dependencies down to one (Docker), solving the configuration issue.

To reduce code rigidity, we heavily follow the software separation of concerns design principle [30]. This ensures that each container/module serves a specific purpose, and changes to any container/module would have little to no cascading effects to the rest of the system. This leads to the creation of four distinct docker containers as highlighted in Figure 10. In combination, these four containers provide the applications front end, back end, database, and TLS/SSL certification services, forming a Mongo Db, Express, ReactJs, and NodeJs (MERN) stack [31]. Docker-compose is used to define and run this multicontainer application, allowing us to define internal networks, shared storage volumes, health monitoring, and build/startup/tear-down procedures. This tool is a huge time saver in the development process by providing a predefined startup script and deployment environment. With each Docker image able to be built separately, docker compose allows the deployed containers to be swapped with new images on the fly. This allows for updates to be rolled out during signup events, without compromising the state of the application.

4.1.2. Containers

Digging deeper into the signup engine shown in Figure 11, the webserver serves as the front end of the application, with responsibility for providing the User Interface (UI) of the web application, as well as routing all incoming network traffic. The webserver is built using Docker, NGINX, yarn, Node(.js), React, JavaScript, HTML, and CSS. React is chosen due to its functional nature and its high capability in dynamically rendering webpages. A functional language allows us to create modular components that can be dynamically rendered by function calls and managed by React’s internal state. Its high capability allows for minimal constraints when looking to add features (for example, the two-factor page). Node(.js) is used for the runtime environment in conjunction with yarn as the package manager. Node(.js) is chosen due to its easy integration into docker, and yarn is chosen due to its higher speed and better security over its competitor npm.
The api-server serves as the back end of the application. It establishes database connections/operations, API calls, and any other complex features needed. The back end forms multiple RESTful APIs which interact with the front end over the internal docker network. These APIs provide the login, signup, two-factor, and dynamic phone number assignment features. This is accomplished through the back-end internal network connection to the Mongo database, deployed in its own Docker container.

The Certbot container handles obtaining and validating HTTPS/TLS certificates by issuing a request to letsencrypt.org to obtain a certificate. For this to happen, the servers at letsencrypt.org need to be able to download files from our server. To allow this, port 80 (http) is opened to all networks in contrast to port 443 (https), which is only accessible via the VT network.

To combat the security risk introduced by leaving port 80 open, we transition to NGINX as the front-end webserver in lieu of Node(js). NGINX functions as a reverse-proxy webserver, allowing us to control the flow of traffic into our application. Hence,
the webserver is configured to allow traffic, on port 80, routed to the certificate challenge endpoint through while redirecting all other traffic to port 443.

4.1.3. Implementation

The webserver is comprised of the root index.jsx/App.jsx files along with four modules: API, component, CSS, and pages. The index/App page wraps the application, which manages the state of the app. The application file provides the navigation bar functionality as well as all functions to update/maintain the state. From the application page, the user can navigate to any page that is defined in the pages module. Each of these pages is comprised of UI functions defined in the components module and styled according to the imported CSS modules. With this setup, we are easily able to add more pages to our application by adding a new file to the pages directory. Our API module is fashioned in a similar way. We use AXIOS to define a simple API interface, enabling us to easily define new API interactions while being scalable.

During the development process, we had to perform a technique called “lifting the state”. We originally had each page maintain its own state, in an attempt to reduce dependencies. However, when a higher state (the Application state) updated to render a different page, the state of the page was lost. This translated to losing all data the user entered on a page when navigating away. A key feature needed was for the user to be able to navigate to the fakeIdentity page, the survey page, and the two-factor page without losing data. Hence, we had to lift the state of all pages out of their React component, localizing all into one centralized state manager. This allowed the user to freely navigate the page without the loss of any data. However, children of a component are not able to update a parent state, a very necessary function to allow the user to interact with the application. This caused the application file to be bloated by update functions which contained permissions to update the state. These were passed down to each page, enabling them to interact with the state. A majority of the bloat was able to be reduced by intelligently forming generic update functions, but we were not able to fully modularize the application.

The api-server is defined similarly by an index.js file along with routes, models, and controllers. The index.js file is run on start, defining and running all api features. It also loads in the database configuration file, setting up a connection on start. It then uses Express(.js) to start a server and listen for requests. Each API is comprised of a route file and a controller file. The route file routes endpoint traffic to its corresponding function in the controller. The controller supplies all functionality for the API through its functions. Currently, each API manages the interactions for each collection in our MongoDB (mainly CRUD operations). Each collection has a corresponding schema, which are all defined in the models module. These models inform our controllers on the format of the collection it is interacting with, allowing us further control in maintaining data integrity.

4.1.4. Two-Factor Authentication (2FA)

In order to provide users with the two-factor data needed to sign up, we need to send all incoming email data to the signup engine. However, these email data might be on a different machine, and the internal processing of its files can be slow, creating a bottleneck. This is accomplished by setting up a webhook. We configure our email server to send a copy of each email received during its initial processing to the web hook. Then, every minute, the api-server requests every item sent to the webhook. Each of these items is prepossessed to extract relevant information, then inserted into our internal MongoDB instance. Once the api-server verifies that it has successfully added the email to its database, it deletes it from the webhook. Via the two-factor API, the front end is then able to request these emails. Working in conjunction with the Users API, we are able to specifically send the emails belonging to the set of fake identities the current logged in user has signed up. This is possible due to the user’s API maintaining the set of all fake identities that the user has signed up with. Using this signup process, we successfully accelerate the process such that individual users can perform 200–500 signups per day.
5. Conclusions and Future Work

As part of a broader effort to track how personal information is shared by online entities, this paper has performed a deep dive into a novel framework that is being used to perform the first level of active OSINT collection. This same framework can be used in tandem with traditional passive OSINT methods like web scraping, traffic monitoring, or the inter-relationships of users to build richer results. Recent and ongoing experiments using the U&A framework include a test of AI-generated papers submitted to publishers listed on Beall’s list and a large-scale test of the 2024 U.S. election cycle, which have incrementally fine-tuned the fake ID generation, signup, and collection processes with nearly 10K identities. Our preliminary results show that collecting information through fake online accounts organized by a single source has distinct advantages over what can normally be discovered in the public domain. This mode does present some challenges, however, as countermeasures to prevent fake account activity online continue to be developed and implemented. Certain characteristics tend to flag fake accounts as they are created or used, with the greatest observed indicators being the lack of immediate activity by the fake identity after account signup. Most such models are trained to detect bots at scale from a malicious source, inferred by activity, and dormant accounts are one of the dominant remaining giveaways after a successful signup. The U&A project is unique both in terms of its scale and robust data-gathering method, utilizing semi-automated fake accounts to track complicated data-sharing behaviors.

Although the fake identities described so far are rich in their documentable attributes, the primary remaining challenge is to ensure that each of the accounts can pass long-term scrutiny. To truly refine the identities and achieve the intent of eliciting content, our fake IDs must have the ability to interact in a convincing manner. The core of this capability is an Account Interaction Engine that selectively chooses, based on the identity, to respond, click links, or open attachments (all within secure virtual machine infrastructures). At the simplest level, the fake ID may be assigned a profile according to the degree to which they interact with externally received content, but it is anticipated that greater flexibility will need to be constructed to avoid account monitors seeking to identify false behaviors [32,33]. A preliminary architecture for an account interaction engine, which is the focus of our current research, is highlighted in Figure 12.

Key elements of this planned architecture include a real-time large language model (LLM)-based email response utility, virtual machine (VM) tools enabling malware detection and the blind clicking of hyperlinks, controlled responses of tracking pixels, and improved user dashboards. This future work will enable real-world interactions, yet will also lead to additional ethical considerations, where the potential burden of an assigned account now goes beyond the one-time signup.

Two final capabilities planned for incorporation into the U&A infrastructure include (1) the dynamic assignment and ability to spoof IP addresses of fake accounts, and (2) the integration of dark web interaction tools. For (1), researchers must be able to modify the location and IP address for each identity. As shown in [26], many detection systems remove accounts that are detected to be registered from the same IP address or location simultaneously, which can affect the quality of data collected. Therefore, a VPN or similar solution will be employed at signup and potentially in any followup interactions. For (2), given that early use-and-abuse experiments demonstrate very little PI sharing by established publicly recognized organizations, we anticipate the need for deeper tendrils in the Internet to identify the origins of online transactions that lead to spam and malware. Early prototyping with Tor and cloud-hosted tools by Bluestone Analytics show promise, yet also highlight a different engagement model, where anonymity is the default rather than PI exchanges for access.
Figure 12. Hierarchical framework for U&A account interaction engine.


Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data from these privacy experiments, including the fake identities, records of which fake identities were used to make one-time transactions, and all content received (email, phone, and SMS text) is planned for release in 2026.

Acknowledgments: This work was supported in part by the Commonwealth Cyber Initiative, an investment in the advancement of cyber R&D, innovation, and workforce development. For more information about CCI, visit www.cyberinitiative.org. Additional support was also received from the VT National Security Institute’s Spectrum Dominance Division, Raytheon Technologies, and ClearlyIP.

Conflicts of Interest: The authors declare no conflicts of interest.

References


Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.