Cybersecurity Across the DNA-Digital Boundary: DNA Samples to Genomic Data

Peter Ney[1], Arkaprabha Bhattacharya[1], Luis Ceze[1], Karl Koscher[1], Tadayoshi Kohno[1], Jeff Nivala[1,2]

1. Paul G. Allen School of Computer Science & Engineering. University of Washington; Seattle, Washington, USA.
2. Molecular Engineering and Sciences Institute. University of Washington; Seattle, Washington, USA.

Abstract

Technological advances in biotechnology, especially next-generation DNA sequencing and direct-to-consumer genotyping, have created exponentially more biological data. To reach this scale, biotechnology pipelines have increasingly relied on automation and computation in the molecular data-processing workflow: biological samples are processed at scale using robotic equipment; molecular sensors, like DNA sequencers, have become specialized computers with peripheral sensors designed to read molecules; and extensive data processing and digital storage is required to manage and make use of this data. All of this computation raises security issues that are more typically associated with computer systems. Here, we explore how the entire DNA data-processing workflow, from physical sample processing through reading DNA into digital information and eventual data analysis, is plagued by a number of security vulnerabilities, including a lack of data integrity, poor software security practices, and hardware that is insecure by design. In standard DNA sequencing pipelines, DNA samples are presumed to be derived from natural sources without manipulation. In this work we show how simple synthetic DNA constructs can be used as vectors for computer malware or as commands to backdoored software or firmware, enabling communication across air gaps. DNA sequencing hardware, including flow cells are also vulnerable by design to data recovery and corruption attacks. Finally, we show how a lack of data integrity checks in genetic databases can lead to catastrophic data breaches and other security concerns. We conclude with some broader themes and lessons from this work that apply to the larger cyber-bio security domain.

Introduction

The biotechnology sector along with companion fields like synthetic biology and bioengineering have raised dual-use security concerns due to the possibility that technologies, such as DNA synthesis or gain-of-function research, could be used maliciously to produce dangerous compounds or organisms. Traditionally biosecurity has relied on security through obfuscation, restricting access or expertise, and high costs as barriers to limit risk. In recent years the biotechnology domain has increasingly relied on automation and computers more generally to accelerate the pace of discovery, improve scalability, and to lower costs. This has greatly expanded the possible attack surface to now include cybersecurity issues such as remote compromise of robotic equipment and laboratory management systems, data and IP theft, and the bypass of other security controls. The old biosecurity paradigm of relying on obfuscation and expertise is no longer sufficient to handle emerging cybersecurity problems, which are much easier to target and require a more active security mindset. This is especially true as unhardened laboratory equipment becomes connected into computer networks.

Together the biosecurity and cybersecurity risks, which we broadly refer to as cyber-bio security (CBS), are more than the sum of their parts. In particular, we have concerns that cyber-vulnerabilities may enhance existing biosecurity concerns — for example, compromised machines being used to bypass security controls to permit the production of dangerous compounds — or that biological material can even work backwards to cause cybersecurity problems. Grappling with the scale of these challenges is difficult because existing biotechnology pipelines are integrated workflows containing many phases ranging from molecular synthesis, sample processing, assay automation and orchestration, molecular sensing, data collection and storage, and finally analysis. In some cases the molecular-digital dichotomy is explicitly blurred like in the case of molecular information storage systems that use molecules to store digital information. To have a better sense of how CBS problems can manifest in real biotechnology pipelines, we focus on the protocols and procedures used with end-to-end DNA processing.

DNA has been read using methods like Sanger sequencing for close to 50 years, but the modern era of reading DNA began with the development of high-density genotyping arrays and high-throughput “next-generation” DNA sequencers, which exponentially increased the scale of DNA analysis. Reading DNA at scale involves computers in all phases of the workflow: biological specimens are prepared for reading using automated assays with liquid handlers, sequencers and microarray instruments are computers with specialized flow cell attachments and cameras, raw data needs to be preprocessed into a usable form (e.g., DNA alignment or assembly), and finally the digital DNA data needs to be analyzed specific to the desired application and stored. Since scalable DNA reading is already used in a wide variety of applications like medicine, genomics, forensics, and consumer testing, a thorough CBS analysis can give insight into risks that may develop in other, less polished emerging technologies and provide lessons for the broader industry.

To see how security manifests in existing DNA reading and processing pipelines, we discuss in the following sections how novel and unexpected CBS problems can appear at almost every phase of DNA processing. First, we demonstrate how DNA's role as a reliable information carrier can be leveraged to encode malicious information directly into synthetic DNA. Maliciously designed DNA can be used to target vulnerable computers downstream of sequencing and even be used as a covert communication channel that bypasses air-gaps to send information to backdoors or trojaned software. Next, we consider how hardware components in sequencing instruments, such as sequencing flow cells, have important implications for secure data deletion or in multi-user environments. Lastly, we discuss the role that data integrity (or lack thereof) plays in securing genotyping data, especially when that data is used in consumer facing applications like genetic genealogy.

DNA as a Malicious Information Carrier

When viewed abstractly, DNA sequencing is the conversion of data encoded physically (in DNA molecules) into a digital form suitable for computer storage and analysis. In the vast majority of cases the information is derived from biological sources like genomes or readouts of biological processes. However, it is possible to artificially synthesize DNA *de novo* using chemical processes (e.g., oligonucleotide synthesis). This means that DNA used in sequencing cannot be presumed to have natural origin. It is the potential for wholly artificial data that creates novel CBS threats to the DNA sequencing pipeline.

Since the early days of cybersecurity, software that insecurely processes input has been a major source of cybersecurity problems [1]. Memory vulnerabilities, like buffer overflows, can allow an adversary to execute malicious code on remote machines giving them full control. In traditional computing, many of these vulnerabilities were latent and only became a significant problem once computers became widely networked and Internet accessible. The lesson here is that it is the existence of software vulnerabilities in combination with the ability of adversaries to target and send data to vulnerable systems that lead to problems.

In many ways DNA sequencing shows similarities to the early days of cybersecurity. As we will show later, widely used bioinformatics utilities show many signs of insecure software design including the use of memory unsafe languages like C/C++, improperly checked memory buffers, and the use of deprecated function calls [2]. In particular, the parts of the software that process the DNA data itself have not been hardened, and the data they process is presumed trustworthy. Once DNA has been sequenced and the sequencing data is being processed by software it is represented in some data encoding and analyzed like any other form of data. This raises the possibility that synthetic DNA could be intentionally designed to encode malicious information including exploitable computer code that could target any vulnerability or be used as a method to send information covertly to software backdoors running on sequencers.

To more concretely understand the risks of synthetic DNA being used as a malicious information carrier we developed two prototype demonstrations. The first example is a simple bioinformatics utility that reads raw sequencing information, early in the sequencing data processing pipeline, and contains an intentionally inserted buffer overflow vulnerability, similar to what is routinely found in unhardened software. The objective was to design a synthetic DNA construct that after sequencing with a modern Illumina sequencer (NextSeq 500) would be able to compromise the vulnerable software and give an adversary full remote control [2]. The second example is a backdoor program we developed that, if covertly run on a sequencer, would be able to decode and respond to commands received from specially crafted DNA molecules. The backdoor was designed so that synthetic DNA messages could be spiked into typical DNA samples to make them covert. An adversary could use this functionality to communicate across air gaps and even exfiltrate data via the resulting sequencing data.

Encoding Malware into DNA

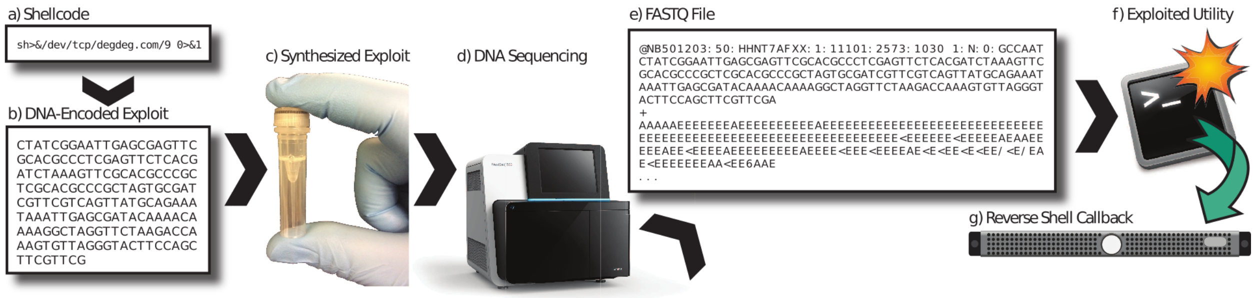
The modern DNA sequencing process (often called next-generation DNA sequencing) happens in three phases: first a DNA sample is prepared for sequencing using standard wet lab assays, then the sample is run through a sequencer which reads the linear sequence of bases in each DNA molecule and stores the raw sequences digitally, and finally, the sequenced data is processed into a usable form with bioinformatics software. Next-generation DNA sequencers cannot read long DNA strands and are limited to reading short strands no more than a few hundred bases in length (each of these small sequences is called a *read)*. Typical genomic DNA can be hundreds of thousands of bases in length, so to accommodate the size limitation, during the sample preparation phase DNA is *fragmented* into smaller pieces via mechanical shearing. Thus, to sequence longer strands, sequencers actually break the strands into small pieces that are read by the sequencer in random order. If necessary for analysis, these random reads produced by the sequencer can be reconstructed into the original, longer sequence. For example, to determine whether a person has a given genetic trait (so called variant calling) the raw DNA reads will need to be cleaned up for quality control, aligned to a reference human genome sequence (effectively ordering the strands), and individual bases determined at each genomic position. This method of pipelining different small utilities together — each with a distinct purpose — to process the sequencing reads in stages is typical in sequencing analysis, and this design makes securing a bioinformatics workflow difficult because each program may be written by different authors and not well supported.

Our objective was to understand what challenges an adversary would face when trying to synthesize and sequence DNA-encoded malware so we could better understand the feasibility of DNA as a malicious attack vector. Our goal was not to identify and target actual vulnerabilities in bioinformatics software. (Although, as described later, the DNA processing pipeline does have especially antiquated software security practices.) Therefore, we begin our study assuming we have already identified a buffer overflow vulnerability in a piece of bioinformatics software. In this case, we took an open source tool written in C designed to compress raw sequencing data files and modified a memory buffer to create a classic buffer overflow vulnerability. Buffer overflow vulnerabilities are a common class of software insecurity that allow adversaries to run their own software on victim computers by sending malformed data. Our goal was to design DNA malware that could be made physically using a low cost DNA synthesis service and survive the sample preparation and sequencing process as intact malware. Any Internet connected machine that reads the malicious sequencing data using the compression utility would be compromised and give an adversary remote access and control.

Our first attempts at designing DNA-malware ran into a number of roadblocks due to DNA synthesis limitations, randomness inherent in next-generation sequencing, and challenges with the DNA encoding scheme used by the vulnerable software. Buffer overflow malware will imbed machine code (called shellcode) and other computer instructions like memory addresses inside the corrupted data. Since the data being sent into the vulnerable utility is raw sequencing data, the malware must be written into the standard DNA bases (A, C, G, and T). The way the bases are encoded when inside the software determines how the shellcode must be written to become functional computer code; in this software each base was encoded using two-bits (A - 00, C - 01, G - 10, T -11). This creates two immediate issues: (1) standard malware when translated into a two-bit DNA encoding scheme results in DNA strands that are difficult to synthesize, and (2) even if the shellcode can be synthesized into DNA, it is unlikely to be read by the sequencer in a way that will result in functional malware.

There were three main synthesis limitations we encountered when we attempted to encode standard shellcode into two-bit DNA. The first was an excessive number or repeated bases which are difficult to synthesize; typical synthesis services limit repeated bases to 10 bases or less. DNA repeats results from shellcode because it is normally repetitive and contains memory pointers with long stretches of 0s or 1s. The second issue was skewed GC-content, the ratio of G/C to A/T that has to be relatively balanced for stable, easy to synthesize DNA molecules. Finally, the repetitive property of shellcode would result in secondary structures in the synthesized DNA — single stranded DNA folding in on itself — again due to common repeating patterns.

Even if shellcode could be synthesized it would not function as intended after sequencing. The shellcode needs to be very small to fit within the length of a single synthesizable DNA (approximately 100-300 bases) or else be stitched back together later down in the data processing pipeline. Another problem was randomness and error tolerance: many reads contain miscalls (incorrectly read bases) that would alter computer instructions, and you cannot predict in advance which direction a strand will be read, which means the shellcode could be reversed.



```
sh>&/dev/tcp/degdeg.com/9 0>&1
```

```
CTATCGGAATTGAGCGAGTTC
GCACGCCCTCGAGTTCTCACG
ATCTAAAGTTCGCACGCCCGC
TCGCACGCCCGCTAGTGCGAT
CGTTCGTCAGTTATGCAGAAA
TAAATTGAGCGATACAAAACA
AAAGGCTAGGTTCTAAGACCA
AAGTGTTAGGGTACTTCCAGC
TTCGTTCG
```

```
@NB501203: 50: HHNT7AFXX: 1: 11101: 2573: 1030  1: N: 0: GCCAAT
CTATCGGAATTGAGCGAGTTCGCACGCCCTCGAGTTCTCACGATCTAAAGTTC
GCACGCCCGCTCGCACGCCCGCTAGTGCGATCGTTCGTCAGTTATGCAGAAAT
AAATTGAGCGATACAAAACAAAAGGCTAGGTTCTAAGACCAAAGTGTTAGGGT
ACTTCCAGCTTCGTTCGA
+
AAAAAEEEEEEEAEEEEEEEEEEEAEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE<EEEEEE<EEEEEAEAAEE
EEEAEE<EEEEAEEEEEEEEEAEEEE<EEE<EEEEAE<E<EE<E<EE/ <E/ EA
E<EEEEEEEAA<EE6AAE
. . .
```

Figure 1. *End-to-end exploit of a computer using DNA.* The shellcode was designed, converted into DNA, synthesized, sequenced, and processed by the vulnerable software. After execution the machine was compromised via a reverse shell.

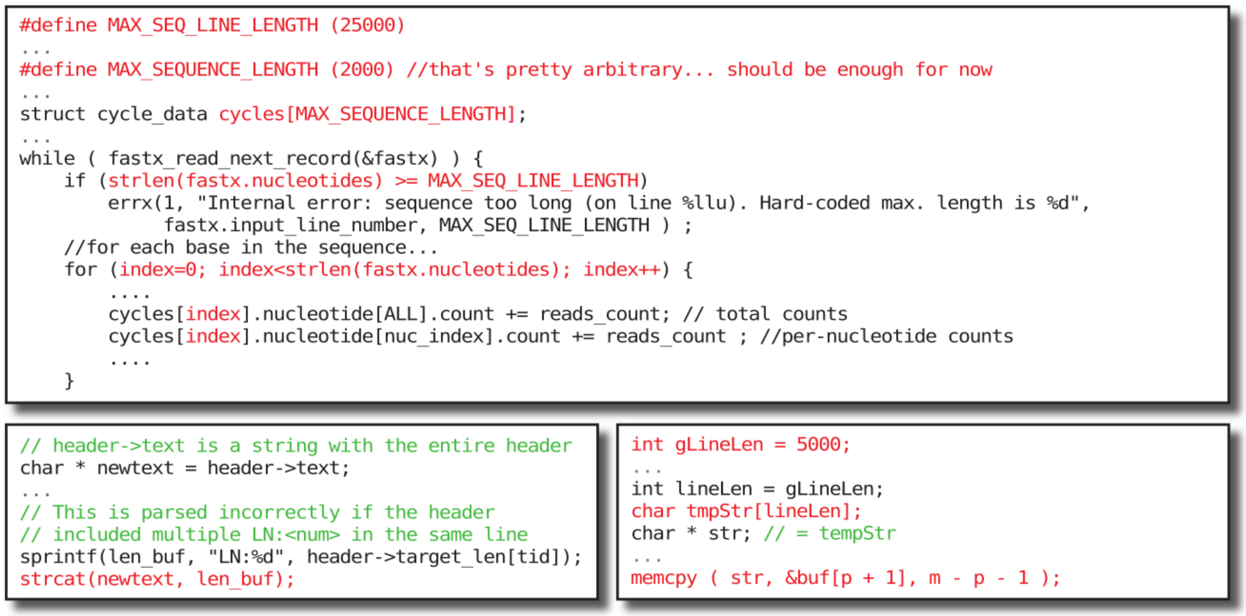
After repeated design and testing we were able to construct shellcode that could successfully navigate the synthesis and sequencing limitations; however, this shellcode was much smaller and less robust than shellcode and malware seen in more realistic scenarios. The shellcode strand was ordered using a commercial synthesis service, sequenced using an Illumina NextSeq 500, and run through the vulnerable utility program. The DNA malware compromised the machine which ran the software and gave us full remote code execution via a reverse shell. (See Figure 1 for a graphical overview of the process.)

While we were able to construct an end-to-end exploit in DNA, the difficulty we had making functional malware for such a rudimentary example means that DNA-encoded malware is not an active concern in most sequencing applications. Yet, as we discuss next, the state of bioinformatics software security is quite poor and emerging technological trends and use cases in biotechnology make this an important vector to study in the future. Many of the constraints we faced, like short read length, are changing with newer sequencing technologies (e.g., long-read Nanopore sequencers) and may make this type of attack more practical in the future. This demonstration highlights how all information vectors into computer systems, including those from physical molecules themselves, should be considered when analyzing the overall security of a system.

Insecurity of Bioinformatics Software

Here, we shift our attention from future looking threats, like DNA-encoded malware, to the more immediate issue of software security in bioinformatics software. DNA sequencing is still an emerging technology domain and so much of the popular software, including software used in commercial applications, is written or maintained by small research groups as open source projects. Much of this software is written in less secure programming languages like C/C++, which are known sources of major security vulnerabilities. Since bioinformatics is not usually considered a cyber security risk, compared to web servers or databases, we expect that bioinformatics software developers have less incentive to write high security software.

To quantify this intuition, we evaluated the software security practices in 13 common sequencing applications used throughout the sequencing workflow, including several present on sequencers. (See [2] for the specific bioinformatics programs and versions we evaluated as of 2017.) All 13 of the programs were open source and written in C/C++. To create a baseline control we also evaluated other popular open source software written in C/C++ that we expected to be under adversarial pressure, like Internet accessible server software. We ran both groups of software through static analysis tools to see if there was a difference in secure software practices and to identify any vulnerabilities. Compared to the controls, the sequencing software had an 11-fold increase in insecure function calls; these function calls are software libraries that have been deprecated because they have known security problems.



```
#define MAX_SEQ_LINE_LENGTH (25000)
...
#define MAX_SEQUENCE_LENGTH (2000) //that's pretty arbitrary... should be enough for now
...
struct cycle_data cycles[MAX_SEQUENCE_LENGTH];
...
while ( fastx_read_next_record(&fastx) ) {
    if (strlen(fastx.nucleotides) >= MAX_SEQ_LINE_LENGTH)
        errx(1, "Internal error: sequence too long (on line %llu). Hard-coded max. length is %d",
             fastx.input_line_number, MAX_SEQ_LINE_LENGTH ) ;
    //for each base in the sequence...
    for (index=0; index<strlen(fastx.nucleotides); index++) {
        ....
        cycles[index].nucleotide[ALL].count += reads_count; // total counts
        cycles[index].nucleotide[nuc_index].count += reads_count ; //per-nucleotide counts
        ....
    }
```

```
// header->text is a string with the entire header
char * newtext = header->text;
...
// This is parsed incorrectly if the header
// included multiple LN:<num> in the same line
sprintf(len_buf, "LN:%d", header->target_len[tid]);
strcat(newtext, len_buf);
```

```
int gLineLen = 5000;
...
int lineLen = gLineLen;
char tmpStr[lineLen];
char * str; // = tempStr
...
memcpy ( str, &buf[p + 1], m - p - 1 );
```

Figure 2. *Buffer overflow vulnerabilities in sequencing software.* Code fragments with buffer overflow vulnerabilities in three different next-generation programs: `fastx-toolkit-v0.0.14` (top), `samtools-v1.5` (bottom left), and `SOAPdenovo2-v2.04` (bottom right). Text in red highlights buggy code, and text in green denotes comments we included for clarification.

More significantly, the static analysis tools were able to identify a number of potential buffer overflow vulnerabilities. After manual inspection we were able to confirm that three of these vulnerabilities could be used to crash the software, a strong sign that they may be exploitable by malware (see Figure 2). In some of the code comments, it was clear that the authors recognized that buffer overflows were possible but did not consider them to be an immediate concern. However, we stress that it is important to patch seemingly non-threatening vulnerabilities like these because they can cause problems in the future as the technology changes. Our security analysis was far from exhaustive, and the ease with which we were able to identify significant vulnerabilities suggests that latent security problems are probably common in bioinformatics and sequencing software.

Using Synthetic DNA to Communicate with a Backdoor and Bypass Air Gaps

Using synthetic DNA as a vector for executable computer code is surprising, but this is just another example of the growing trend of using synthetic DNA for non-biological purposes. For example, DNA data storage systems are capable of archiving large digital databases into DNA and even computing with molecules [3]. This highlights how scalable DNA reading and writing — through improvements in sequencing and synthesis — makes DNA an effective universal information carrier and raises the possibility that other forms of information encoded in DNA could be used maliciously. In particular, we are interested in how DNA molecules or DNA sequencing data can be used as a covert means to communicate to a malicious backdoor on sequencers or computers downstream of sequencing.

Backdoors are covert programs, alterations to software, or even physical hardware modifications that are used to bypass typical security controls and give an adversary unauthorized access and control to computer systems. Backdoors have a long history in cybersecurity and are used by a wide variety of actors for different purposes [4]. They are useful in statecraft to gain intelligence on high value targets, as a tool for corporate espionage to steal intellectual property, and as a means to destroy or incapacitate computers and equipment. Hardware backdoors are often placed into systems during manufacturing or upstream in the supply chain, while software backdoors may be inserted directly into compromised computers or even via small modifications to open source software or data. Given the challenge of preventing and detecting covert backdoors, one approach to ensure security in high risk environments is to air-gap any critical systems; air-gapping is a method of isolating machines on a secure network without direct connectivity to wider area networks like the Internet or avoiding networks all together. Air-gapping reduces the cyber attack surface by physically blocking malicious messages from reaching potentially vulnerable machines.

In the context of DNA sequencing, there are a number of reasons to believe that DNA sequencers would be useful targets for backdoors. DNA sequencers process lots of sensitive medical or genetic information and are often used in research facilities to sequence valuable intellectual property, like engineered organisms or therapeutics. Sequencers may also be located in secure networks within wet labs which may give adversaries a foothold to compromise other important systems. Given the cost of high-end sequencers, they may be valuable targets for ransomware or denial-of-service attacks.

Sequencers are essentially regular computers with specialized hardware and software for sequencing — current day Illumina sequencers run Windows 10 and have network capability, mouse and keyboard peripherals, and monitors like conventional desktop computers. Therefore, when sequencing in sensitive circumstances it is recommended that sequencers are only connected to secure networks or disconnected altogether (i.e., air gapped). However, for a sequencer to meet its basic functions it must be able to take in DNA samples and produce sequencing data as output, even if it is otherwise completely isolated. It is this sequencing interface — DNA samples as input and files as output — that creates a possible communication channel between adversaries and backdoors running on (supposedly) air gapped sequencers.

Consider the case where an adversary successfully inserts a backdoor into a DNA sequencer that communicated only through the sequencing interface (i.e., sequencer is air gapped). For this limited backdoor to be useful the adversary would need to be able to direct DNA to a compromised sequencer, and if return communication is necessary (e.g., to exfiltrate data) the adversary would need the resulting sequencing data. Even with these constraints, there are a number of situations today where adversaries can direct DNA to sequencers. Outsourced sequencing facilities allow third parties to submit samples to be sequenced and direct-to-consumer genetics enables anyone the ability to submit DNA samples through the mail to be sequenced or genotyped. (See the later section on data integrity in DNA processing for other cybersecurity issues in the consumer genetics industry.) Other possibilities include insiders with direct access to sequencers, situations where an adversary could anticipate that DNA will be sequenced (like with forensics), and even emerging technologies like DNA data storage systems where users may be able to directly specify 'DNA files' to be read by sequencers. As sequencing, and DNA processing more generally, becomes more ubiquitous we expect these possibilities to grow.

Developing a Backdoor

We developed a prototype backdoor to run inside the Illumina iSeq 100 to better understand the feasibility and challenges of constructing a backdoor that only communicates to an adversary via the sequencing interface. The backdoor was designed to receive commands from covert instructions encoded in DNA strands and write output into the DNA sequencing data. Backdoor messages written into DNA could be stealthily mixed into normal DNA samples (e.g., genomic DNA) and still be parsed and executed; sequencing regular DNA would be unaffected by the backdoor. Once decoded, the messages sent to the backdoor are treated as arbitrary commands that can be executed. Possible commands could include network mapping if the sequencer is on a private network, copying or exfiltrating data present on the sequencer, or even wiping the sequencer to render it unusable. Any output would be covertly encoded back into DNA bases and written into the sequencing output file along with the legitimate sequencing data.

When a DNA sample is processed, all of the separate DNA strands are sequenced together on a piece of hardware called a flow cell. (The flow cell has interesting security properties that will be discussed later.) The sequence of bases (i.e., A, C, G, and T) in each strand are read one at a time with a camera that detects fluorescent light emitted during the sequencing process. This data is eventually written into intermediate binary files called BCL (binary base call files). These BCL files are later converted to plain text files that are more suitable for bioinformatics analysis; however, since these files are the direct output of sequencing and are in an obtuse binary format, they make a good location for the backdoor to inject itself into the sequencing pipeline.

The backdoor functions as follows: After being inserted onto the sequencer it reads the batches of BCL files generated during sequencing. (These are typically placed in a fixed location in the sequencer's file systems so they are easy to locate.) From these BCL files the backdoor extracts any DNA reads meant for the backdoor, decodes those DNA reads back into the original message, executes any commands that are required, and appends any return messages or data back into the same BCL files. Viewed abstractly, this is just a specific way of encoding/decoding arbitrary digital data into and out of DNA, which is what existing DNA data storage pipelines already accomplish. So to make a backdoor, we repurposed existing DNA data storage software to function in a new context. (See Figure 3 for an overview of this process.) While a detailed treatment of the backdoor architecture is beyond the scope of this chapter, we describe the high level principles used by backdoor encoder, decoder, and exfiltration module below.

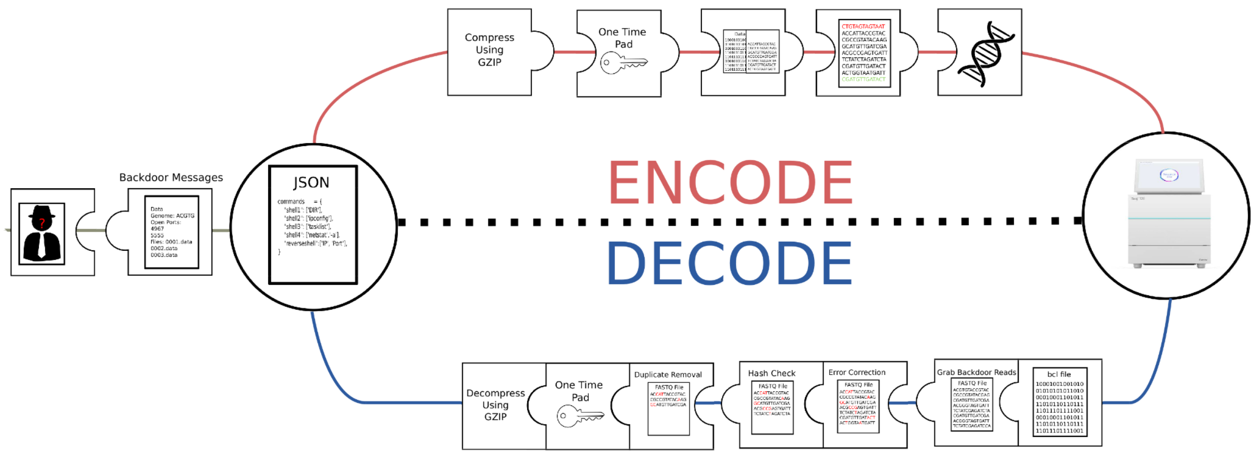


Figure 3. *Communication lifecycle for a backdoor on a DNA sequencer.* Backdoor messages and commands are sent and received from the backdoor as JSON messages. *Encoding*: (1) JSON commands are compressed; (2) data is XORed with a fixed string for 'randomization'; (3) data converted into DNA bases and broken into small chunks; (4) strands are appended with a tag and hash; and (5) reads are synthesized into physical DNA. *Decoding*: (1) BCL files are parsed and message reads removed; (2) reads are checked for proper hash and errors corrected; and (3) duplicates are removed, the 'randomization' is reversed, and the data is uncompressed into the JSON commands.

Encoding Module

The encoding module takes data messages as input — text based JSON messages — and outputs a set of DNA sequences representing the input message. To be an effective encoder for this backdoor it needs to meet three requirements: (1) the output DNA sequences must be possible to synthesize (see the previous section for challenges that can occur with DNA synthesis), (2) each output strand must be small enough to fit within a single read (approximately 100-200 bases), and (3) there needs to be some tag included so that the backdoor can distinguish message reads from other DNA strands, such as genomic DNA, that might be mixed in. Since sequencers read DNA strands in no particular order, this means that output strands must be designed in a way so that larger messages can be broken into pieces and later reconstructed.

To encode a message it is sent through the following pipeline: First, desired commands are written into a text-based JSON object and compressed using gzip to reduce the message size. Next, the compressed message is XORed with a fixed string to fully 'randomize' the message. This is necessary because random-like DNA bases with this encoding are more easily synthesized because it makes repeats and GC-content issues less likely. The randomized data is then broken into fixed length, 18-byte chunks so it can fit within a single short strand, with each chunk assigned a 3-byte index used to signify the strand ordering. The binary-to-DNA encoding scheme we used comes from an existing DNA data storage encoding scheme [5]. After encoding, each chunk+index (denoted as the *payload)* fits in 96 DNA bases. Finally, we include two last pieces of information in each strand: a fixed tag to signify the strand as designated for the backdoor and a hash of the payload that can be used to filter out reads with errors. At this point the adversary has a set of strands, each 152 bases in length, that encode the intended message and are ready for synthesis.

Decoder Module

Decoding is roughly the reverse of the encoding process with some important differences. The decoder takes BCL files produced by a sequencer and returns the decoded message or instructions to be run by the backdoor. The decoder is filtering for messages with the correct tag prefix in the BCL files, filtering reads with errors (determined via the hash), converting the DNA bases into binary data, and placing them in the correct order to produce the final JSON message. Each specific strand will be present in multiple copies, which is helpful for redundancy; these duplicates also need to be filtered out.

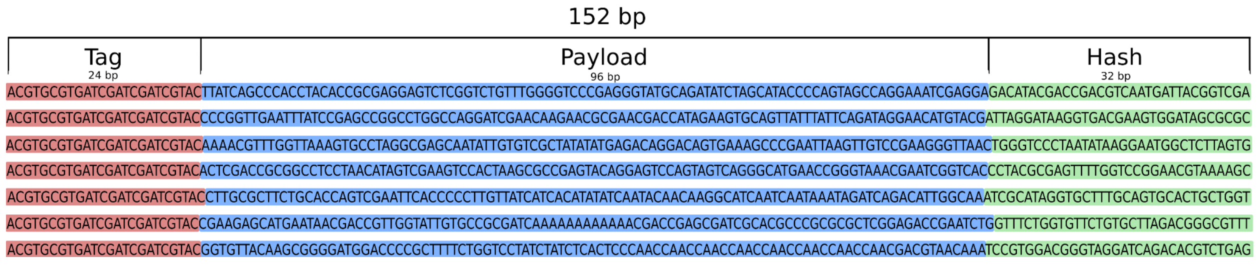
Exfiltration Module

Some communications to the backdoor do not require a response. For example, if the goal is denial-of-service (DoS), the destruction of data, or as a means to send malware, the command may not require a response. However, we were also interested in cases when the adversary wants to receive messages back from the backdoored sequencer. In some cases the sequencer may be networked and so it can provide a response back over the Internet, but to return a message via the sequencing interface the response would have to be returned to the adversary in the sequencing data files (e.g., at an outsourced sequencing facility). To accomplish this, we encoded the response into DNA using the encoder module and appended the data to the end of the BCL files produced by the sequencing run. After receiving the BCL files the adversary would just use the decoder as usual to parse the response.

Prototype

We developed a prototype backdoor designed for the Illumina iSeq in 800 lines of python code implementing the encoder, decoder, and exfiltration module described above. To keep this simple, we built it as a Windows 10 application to be run on the sequencer that would be pointed to the BCL files; however, in realistic conditions the backdoor could be designed more covertly, for example by including it in the OS or embedded in the sequencing software.

To demonstrate end-to-end functionality of the backdoor, we encoded 5 commands into a single JSON message that included: network scanning, reverse shell, two OS scanning commands, and an instruction to exfiltrate data off the sequencer. When encoded, these commands could fit within DNA strands (see Figure 4) that were ordered using a commercial synthesis service. After sequencing, the strands were properly decoded and executed, and the exfiltrated data was appended to the BCL files.



```
ACGTGCGTGATCGATCGATCGTACTTATCAGCCCACCTACACCGCGAGGAGTCTCGGTCTGTTTGGGGTCCCGAGGGTATGCAGATATCTAGCATACCCCAGTAGCCAGGAAATCGAGGAGACATACGACCGACGTCAATGATTACGGTCGA
ACGTGCGTGATCGATCGATCGTACCCCGGTTGAATTTATCCGAGCCGGCCTGGCCAGGATCGAACAAGAACGCGAACGACCATAGAAGTGCAGTTATTTATTCAGATAGGAACATGTACGATTAGGATAAGGTGACGAAGTGGATAGCGCGC
ACGTGCGTGATCGATCGATCGTACAAAACGTTTGGTTAAAGTGCCTAGGCGAGCAATATTGTGTCGCTATATATGAGACAGGACAGTGAAAGCCCGAATTAAGTTGTCCGAAGGGTTAACTGGGTCCCTAATATAAGGAATGGCTCTTAGTG
ACGTGCGTGATCGATCGATCGTACACTCGACCGCGGCCTCCTAACATAGTCGAAGTCCACTAAGCGCCGAGTACAGGAGTCCAGTAGTCAGGGCATGAACCGGGTAAACGAATCGGTCACCCTACGCGAGTTTTGGTCCGGAACGTAAAAGC
ACGTGCGTGATCGATCGATCGTACCTTGCGCTTCTGCACCAGTCGAATTCACCCCCTTGTTATCATCACATATATCAATACAACAAGGCATCAATCAATAAATAGATCAGACATTGGCAAATCGCATAGGTGCTTTGCAGTGCACTGCTGGT
ACGTGCGTGATCGATCGATCGTACCGAAGAGCATGAATAACGACCGTTGGTATTGTGCCGCGATCAAAAAAAAAAAACGACCGAGCGATCGCACGCCCGCGCGCTCGGAGACCGAATCTGGTTTCTGGTGTTCTGTGCTTAGACGGGCGTTT
ACGTGCGTGATCGATCGATCGTACGGTGTTACAAGCGGGGATGGACCCCGCTTTTCTGGTCCTATCTATCTCACTCCCAACCAACCAACCAACCAACCAACCAACCAACGACGTAACAAATCCGTGGACGGGTAGGATCAGACACGTCTGAG
```

Figure 4. *Backdoor commands encoded into DNA*. 7-strand sequence encoding a JSON message instructing the backdoor software to scan the network, open a reverse shell to the adversary, run OS scans, and to exfiltrate data on the sequencer into sequencing data files.

Cyber-Physical Security and the Molecular-to-Digital Hardware Interface

In this section, we consider the cybersecurity aspects of a different component of the sequencing workflow: the flow cell hardware used to convert physical DNA into digital data [6]. Flow cells are small, disposable fluidics devices contained within sequencers to hold the samples and reagents during sequencing. The enzymatic process of sequencing — called sequencing-by-synthesis in Illumina sequencers — happens on the surface of the flow cell. DNA strands stick to the surface of the flow cell and enzymes, like DNA polymerase, are used to read the DNA in the strands one base at a time; each time a base is added it emits a fluorescent signal that can be read by high-resolution cameras. The flow cell, therefore, is the primary piece of hardware responsible for the molecular to digital conversion.

An important theme in cybersecurity is that interfaces and boundaries between different systems is a common source of security issues. The reason is that engineers may have unspoken assumptions or different mental models about what is possible on either side of the boundary. Regarding flow cells, we suspected that the molecular nature of DNA could lead to unanticipated information security risks because of properties inherent to physical DNA molecules. For example, DNA molecules are stable for long periods at room temperature and can be enzymatically amplified (i.e., copied). This may have implications for unauthorized data recovery because usable information may be recoverable from improperly disposed flow cells. Another security concern has to do with how different DNA samples are mixed together to improve sequencing efficiency. So called multiplex sequencing is a technique whereby DNA samples are combined into one solution, sequenced together, and data from the individual samples are separated out later in software. However, we show how small errors in this process, due to sequencing chemistry and flow cell design, can be leveraged by adversaries to maliciously alter the genetic interpretation of other samples.

Data Remanence

Flow cells used by the most popular Illumina sequencers are meant to be single use and discarded after sequencing. However, there is little guidance on how to properly dispose of flow cells, and so oftentimes flow cells are just thrown into the trash. We hypothesized that discarded flow cells contain enough residual DNA to recover sensitive information from a previous sequencing run. This is related to data remanence attacks against traditional magnetic hard drives; residual representations of data still remain on disks even after file deletion and improperly wiped and discarded hard drives create an information security risk.

We developed a simple protocol to collect the residual DNA stuck to the flow cell by flushing laboratory-grade water multiple times through the flow cell's fluidic channel and collecting the waste water. This waste contains a portion of the residual DNA on the flow cell in solution where it can be amplified (i.e., exponentially copied) using polymerase chain reaction (PCR). The amplified product is resequenced to read out the 'improperly deleted' data. We tested this protocol out using iSeq flow cells on two different DNA inputs: a high redundancy DNA data storage file used to evaluate error rates and file recovery and a human genome sample sequenced at low coverage (low redundancy) to let us explore the limits of data recovery.

DNA data storage files are highly redundant and tolerant to errors, so the file can still be fully recovered when there are missing strands. In this case, 96.5% of the unique strands representing the file were present in the residual sample, which was sufficient to fully reconstruct the file without error. In the human genome sample, which had much lower redundancy, we were able to recover approximately 1.8 million unique DNA reads compared to 4.4 million unique reads in the original sequencing run. This makes for a 40.1% residual recovery rate for low redundancy samples. For most genomic sequencing applications, including medical diagnosis, the 40% yield we recovered from the used flow cell would be sufficient to predict the bases in a person's genome.

These experiments show that, similar to hard drive data remanence problems [7], residual data recovery is possible on discarded flow cells. The security risk of this disclosure will depend on the specific sequencing application, but it is substantial for typical genomics and DNA data storage pipelines. Laboratories should consider flow cell remanence in their sequencing pipelines and simple solutions like the physical destruction of flow cells may prevent unintended information leakage.

Data Leakage Between Samples

High-throughput DNA sequencers improve throughput and reduce per sample sequencing cost by sequencing multiple samples concurrently. This is accomplished using short DNA barcodes (6-8 base strands) that are appended to all DNA strands and made unique for each sample. After sequencing, the barcodes can be used to separate each DNA read into the corresponding file for each sample (called demultiplexing). Barcoding and demultiplexing has a high success rate of assigning DNA reads to the correct sample (over 99.9%). In sequencing runs producing over 100 million reads, less than 1000 reads will be assigned to the incorrect sample. This low level of improper assignment is due to particular flow cell architectures (non-patterned) and unintended enzymatic side-effects during sample preparation. While this level of error is negligible in most routine sequencing applications, it can be utilized by adversaries as a way to alter other samples in a reproducible and specific manner.

To show this, we conducted the following experiment. We began with two DNA samples: one an actual human genome sample and the other a synthetic sample designed to look like the genetic variant responsible for sickle-cell disease (a single base substitution of A to T). The synthetic sample was a short fragment of DNA identical to the wild type human *β-globin* gene, except that it included the T base reflecting sickle-cell trait. This fragment was inexpensively synthesized using a commercial DNA synthesis service. These two samples were sequenced in a multiplex fashion according to the usual protocols. As anticipated, enough DNA encoding the sickle-cell trait leaked from the synthetic sample into the human genome index to cause the human sample to appear like a sickle cell carrier. This is possible because at any given genomic position there is only a small amount of coverage (<200 read depth on average). Therefore, even a small amount of incorrect assignment during demultiplexing (<0.01%) can be sufficient to alter genetic interpretation.

What this simple demonstration shows is that seemingly independent samples, when sequenced together, can lead to undesirable side-effects. When this side-effect is not intentionally misused, its effects are negligible in routine sequencing applications; however, when directed with intelligence, side-effects like this can be used adversarially. While not detailed here (see [6] for details), the synthetic DNA used in the sickle-cell sample can even be spiked into tissue samples like saliva and have a similar effect after sequencing. This type of theoretical attack highlights how sequencing operators should be wary when untrusted samples are sequenced concurrently, especially if those samples can be submitted by consumers.

Data Integrity in DNA Processing

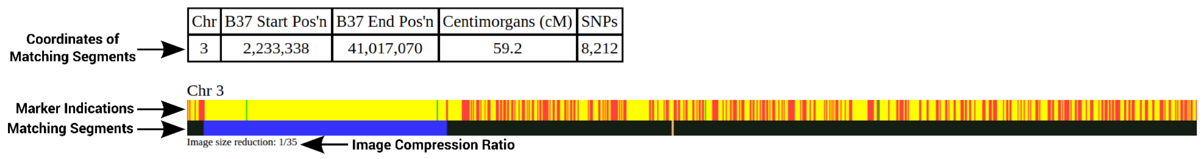
Previously, we have explored CBS aspects of the earlier stages of the DNA processing pipeline. Here, we focus on the security of the last component of DNA processing: data generation, analysis, and storage.

The most mature and widely accessible DNA analysis services come from the direct-to-consumer (DTC) genetic testing industry. DTC companies like 23andMe and AncestryDNA have processed genetic samples from 10s of millions of customers collected through the mail [8]. The vast majority of DTC testing is done using high-density genotyping arrays that measure individual genetic markers (known as

SNPs) at approximately 500,000 locations in the genome. These tests are low cost (<$100) and give customers insights into their health, ethnicity, and other traits. The ability of DTC tests to identify close genetic relatives has been one of the most exciting applications of genetic testing and spawned the field of genetic genealogy, which combines genetic data with existing datasets, like family trees, to identify unknown relatives. Genetic genealogy has proved so successful that it has been co-opted for other uses including as an aid to forensics to identify the source of DNA samples from crime scenes, known as investigative genetic genealogy (IGG).

Data security in genetic genealogy is made especially hard because there is lots of data sharing. Some of the most popular genetic genealogy tools are 3rd party applications that accept genetic uploads from users directly; users are tested with a DTC service, download their raw genetic data, and then upload it to a 3rd party service for analysis. This approach of data sharing raises many security concerns, but we focus on the security risks derived from one problem: the lack of data authentication. We show how this fundamental problem can lead to catastrophic security risks to genetic genealogy services. The lack of authentication gives adversaries the opportunity to steal private user genotypes from genetic databases and upload corrupted results to appear like fake relatives.

In 2019 we studied GEDmatch, the most popular 3rd party genetic genealogy service [9]. GEDmatch runs as a web service that lets users upload files to a central database to run genetic genealogy analysis. GEDmatch is a favorite of genetic genealogists and law enforcement because it is an open platform that gives users fine grained control over queries and returns extensive results and visualizations. Most significant for us is that there is no control over what kind of information can be uploaded to the service. (In fact, when law enforcement uses GEDmatch for IGG, they upload artificially generated files constructed from crime scene samples.) We hypothesized that this design could lead to serious vulnerabilities for a few reasons. First, the fact that users can upload any data, so long as it is formatted like a typical DTC file means that an adversary has significant flexibility over what can be uploaded to GEDmatch, including pathologically designed data. This can be combined with significant user control over what queries can be run to give an adversary a lot of leeway to target any identified vulnerabilities. The results returned by queries also include high-resolution chromosome images of DNA comparisons, a basic technique in genetic genealogy that can reveal a lot of potential sensitive information (see Figure 5 for an example of a chromosome comparison).



Chr	B37 Start Pos'n	B37 End Pos'n	Centimorgans (cM)	SNPs
3	2,233,338	41,017,070	59.2	8,212

Figure 5. *DNA comparison of chromosome 3 between two users on GEDmatch.* Colors indicate the degree of DNA sharing.

To explore how the GEDmatch service architecture could lead to cybersecurity problems we created two users on the service, one representing an adversary and the other a victim. Under the victim user we uploaded 5 genetic profiles constructed from open source data (GEDmatch allows users to upload more than one genetic data file), and to the adversary user we uploaded artificial data designed to attack the victim profiles. We configured the privacy settings of the uploaded files to not interfere with or view any real user data. All vulnerabilities we discovered were disclosed to GEDmatch and patched prior to the publication of our work.

We were first interested how DNA comparisons used to predict ancestry can be used to exfiltrate sensitive genetic data from other users in the database. To predict a relationship, the genetic profiles of two users are compared to find long stretches of chromosomal DNA that are nearly identical (so called matching segments). The closer the relationship, the more matching segments there will be between the two files, and the degree of the relationship can be predicted by the distribution of matching segments. To find unknown relatives, GEDmatch lets users run matching queries between files they owned and any other files in the database. However, the chromosome visualizations returned in these comparisons (Figure 5) leak too much information about how the two files differ from one another.

GEDmatch takes significant steps to obfuscate the underlying genetic data, so to actually take advantage of this vulnerability is an involved process (see [9] for details). However, the high level attack is straightforward. The adversary uploads specially designed artificial data files that return deterministic results depending on which specific genetic markers are present in the other file — the color of the visualization can be analyzed at the pixel level to get SNP level resolution. These "malicious" artificial files can then be compared to any file in the database to reconstruct its private genetic markers, and any missing gaps can be filled in using publicly available genetic information (via imputation). In total, we were able to predict 92.6% of the markers in each of the experimental profiles we uploaded with 98.4% accuracy using this technique. This attack could then be automated to extract data from every file in the database.

The second question was whether the relationships predicted by GEDmatch could be manipulated or forged by an adversary. This could be useful to someone trying to impersonate a lost relative or even as a tool to evade detection in an IGG investigation. Since genetic relationships are found by looking for users with shared DNA segments, any artificial data that matches segments with another user will appear like a new relative. Using an approach similar to the data extraction attack and by taking advantage of a form of compression used by GEDmatch, we were able to generate wholly artificial genetic files that could appear like arbitrary relatives for any user on GEDmatch. In Figure 6 we show two simplified examples where forged relatives can be submitted to a genetic genealogy database to imply incorrect genealogical inferences.

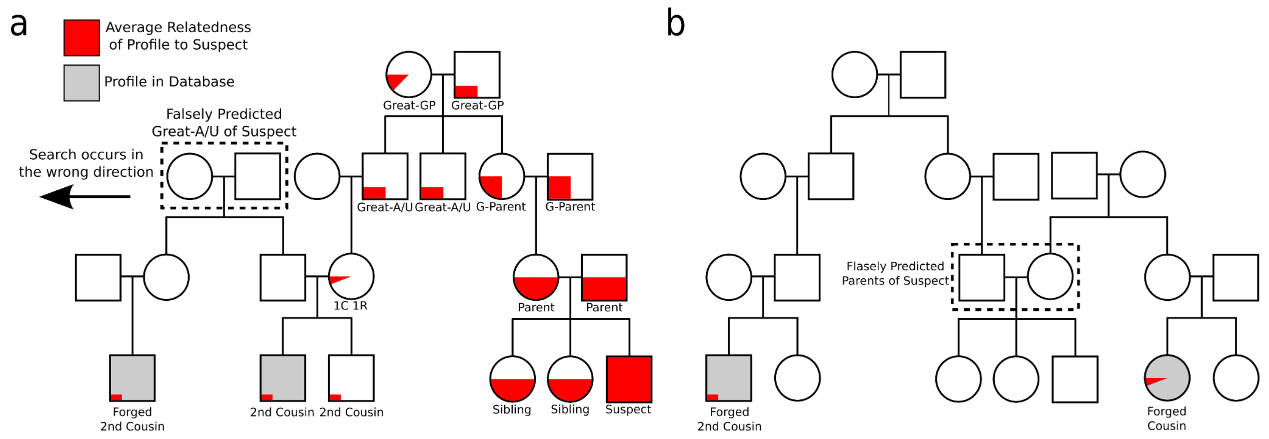


Figure 6. *Example attacks using forged relatives to imply false ancestry.* **a.** An adversary wants to avoid identification when their 2nd cousin is already in a third-party database. The adversary uploads a falsified second cousin under the identity of a second individual that is related to the 2nd cousin but not the adversary. This falsely implies that the adversary is on a different branch of the family tree. **b**. The adversary uploads two falsified relatives on different branches to falsely imply that a couple was the adversary's parents.

The fundamental vulnerability in both of these examples comes from the lack of data authentication. There is no confirmation that uploaded genetic data actually originated from a legitimate DTC service. As long as the data is properly formatted like DTC data, any kind of file can be uploaded. When artificial uploads are allowed on a feature rich service with complex, user driven analysis, major security risks are almost inevitable. GEDmatch relied on techniques like obfuscation and compression, which they believed were sufficient to deter attackers. This type of faulty reasoning is a common theme in cybersecurity; obscurity is not a substitute for rigorous security design principles. For example, digital signatures or direct file transfers from the DTC companies to 3rd party services would have been sufficient to avoid these problems. Moving forward, we believe the lessons for the DTC industry and any field with significant genetic data sharing is that it is essential that there are some assurances about the authenticity or provenance of any data that is shared and analyzed.

Conclusion

In this chapter we performed an extensive security analysis on many stages of the DNA processing pipeline from raw DNA samples to sequencing and eventual storage and analysis. While this is not an exhaustive list of potential issues, these results show how challenging CBS can be when applied to mature technologies. Security in the biotech domain is much more complex than simply layering cybersecurity into a wet lab context because there are unique security dimensions and risks that do not exist in other domains. Like traditional cybersecurity each technology and system will have its own unique security problems. However, there will be common security paradigms and lessons that apply across domains. We conclude this chapter with a few broad CBS takeaways that we hope are useful for biotech engineers to consider as they design and implement new technologies.

Biotechnology has a complex threat surface: Many biotechnology pipelines, like DNA sequencing, are long and complex. They involve different hardware components and take data and commands from many sources. This type of design makes security particularly hard because design choices made in one stage of the process can have effects on seemingly independent stages later on. We saw this many times in the DNA processing pipeline. For example, flow cell design affected data integrity, digital information could be encoded in physical molecules and affect downstream analysis, and the lack of authentication by DTC providers affected user driven data sharing. Any security analysis of a biotech pipeline will need to view the entire process holistically and consider how the process and data may be used unexpectedly.

Pay attention to interfaces and data boundaries: One issue we saw repeatedly were problems at the boundary between different phases or stages of a biotechnological process. For example, places where physical data was converted into a digital form. Issues at interfaces are common in cybersecurity; they occur because there are mismatched assumptions between the designers on either side of a boundary. As with other cyber-physical systems, such as automobiles [10], biotech engineers should pay special attention to any interface or conversion point and make sure the expected behavior of the interface is understood in advance.

Confirming authenticity and integrity are critical for security: It is much harder to build security into a process when engineers do not consider the trustworthiness or validity of any input data or commands. Any part of a biotech process that is downstream of data input, whether physical or digital, should consider: (1) is the source of this data trustworthy and (2) is the data formed as expected. This can be accomplished by regulating access only to trusted parties when possible via authentication and authorization, but when that cannot be assured, data should be sanitized before being processed by software.

Current biotech and bioinformatics software do not follow cybersecurity best practices: Our analysis showed that bioinformatics software does not meet security standards. However, we suspect this problem is much broader across the industry because there are few incentives for secure software design. For example, laboratory management systems, control software, and robotics equipment, like liquid handlers, all have potential security problems. If true, latent software vulnerabilities could exist throughout many current and future biotechnology processes. This is reminiscent of the early stages of computers where latent vulnerabilities were not exploited until computers were later connected. The industry should get ahead of this problem and begin hardening and patching software before problems manifest.

Current trends in biotechnology make security important in the future: Biotech is becoming more automated, integrated with computers, connected, and accessible. This means that CBS problems will only become more feasible over time. While it might be tempting to address security risks through restricting information about and access to biotech systems, such an approach is insufficient. In the biological sector, there is extensive data sharing, the need for connectivity and remote access, and demand by end customers for services. Further, as we demonstrated, even the core input into many bio-systems (like DNA) is arbitrarily writable and creates additional attack surfaces not present in other security domains. Engineers need to take a more active security approach to get ahead of security problems, especially in domains with dual-use biosecurity risks.

Acknowledgements

This research was supported in part by a grant from the DARPA Molecular Informatics Program, NSF Grant CNS-1565252, the University of Washington Tech Policy Lab (which receives support from the William and Flora Hewlett Foundation, the John D. and Catherine T. MacArthur Foundation, Microsoft, the Pierre and Pamela Omidyar Fund at the Silicon Valley Community Foundation), the Short-Dooley Professorship, and the Torode Family Professorship.

References

1. Orman, H. (2003). The Morris worm: A fifteen-year perspective. *IEEE Security & Privacy*, *1*(5), 35-43.
2. Ney, P., Koscher, K., Organick, L., Ceze, L., & Kohno, T. (2017). Computer security, privacy, and DNA sequencing: compromising computers with synthesized DNA, privacy leaks, and more. In *26th USENIX Security Symposium (USENIX Security 17)* (pp. 765-779).
3. Ceze, L., Nivala, J., & Strauss, K. (2019). Molecular digital data storage using DNA. *Nature Reviews Genetics*, *20*(8), 456-466.
4. Adee, S. (2008). The hunt for the kill switch. *IEEE SpEctrum*, *45*(5), 34-39.
5. Takahashi, C. N., Nguyen, B. H., Strauss, K., & Ceze, L. (2019). Demonstration of end-to-end automation of DNA data storage. *Scientific reports*, *9*(1), 1-5.
6. Ney, P., Organick, L., Nivala, J., Ceze, L., & Kohno, T. (2021). DNA Sequencing Flow Cells and the Security of the Molecular-Digital Interface. *Proceedings on Privacy Enhancing Technologies*, *2021*(3), 413-432.
7. Peter Gutmann. Secure Deletion of Data from Magnetic and Solid-State Memory. In Proceedings of the 6th USENIX Security Symposium, San Jose, CA, volume 14, pages 77–89, 1996.
8. A. Regalado, “More than 26 million people have taken an at-home ancestry test,” MIT Technology Review, 2019.
9. Ney, P., Ceze, L., & Kohno, T. (2020). Genotype Extraction and False Relative Attacks: Security Risks to Third-Party Genetic Genealogy Services Beyond Identity Inference. In *Network and Distributed System Symposium (NDSS 2020)*.
10. Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., ... & Kohno, T. (2011). Comprehensive experimental analyses of automotive attack surfaces. In the 20th USENIX Security Symposium (USENIX Security 11).