Enriching Attack Trees by Reconstructing the Equifax Data Breach

STEFAN MORRIËN, University of Twente, The Netherlands

Attack trees and crime scripts are frequently employed in cybersecurity to analyze vulnerabilities from the perspective of a criminal. However, no development has been made in researching whether these two formalisms can complement each other. We propose that a detailed attack tree can be generated by recreating the steps outlined in a crime script. Our objective is to uncover hidden attack vectors represented as nodes that were not yet in the tree and improve the completeness of the attack tree in terms of the number of nodes (tree depth), using the 2017 Equifax data breach as the main focus. This was done by simulating the Equifax environment on a virtual machine in which a vulnerable version of Apache Struts was running. The attack was then recreated by following the steps of a crime script, and any missing steps were analyzed, along with alternative paths that the attacker could have taken. Based on these results, a detailed attack tree was created that contained the attack in significantly more detail than the initial attack tree created from the crime script and also included alternative paths.

Additional Key Words and Phrases: Attack Trees, Crime Scripts, Equifax, Data Breach, Apache Struts

1 INTRODUCTION

In cybersecurity, understanding vulnerabilities and how they are exploited by attackers in a system is vital for building secure systems. Adopting the perspective of an attacker can often reveal vulnerabilities that a security specialist might have overlooked. They also provide insight into which attacks are the most likely to occur. This perspective is also valuable when researching data breaches, such as the Equifax data breach in 2017 [13]. This is one of the most impactful data breaches, resulting in the exposure of important private information, including names, Social Security numbers, dates of birth, addresses, driver's license numbers, credit card numbers, and dispute documents of 148 million people—nearly half the U.S. population. There has also been some research on this breach [13, 18, 20].

This paper aims to analyze this breach using attack trees [17] and crime scripts [6] to understand how the latter can complement the former, which has not been done in other papers. This will be achieved by attempting to convert a crime script into an attack tree. Crime scripts and attack trees both provide valuable insights into cyberattacks; however, they serve different purposes. The purpose of attack trees is to analyze all the paths an attacker could take to achieve their goal: exfiltrating valuable data from the systems of Equifax. However, a crime script describes a single attack in great detail, including how it happened, what steps the attacker took, and what prerequisites were necessary. Combining these two can significantly enrich security analyses and uncover hidden attack vectors that traditional methods, such as using only attack trees or crime scripts, might overlook. By bridging these two approaches, a deeper understanding can be gained of both the attacker's tactics and the overall security weaknesses of a system due to the exposure of hidden attack vectors.

Preparation
On the 7th of March 2017, a remote command injection (RCI) vulnerability found in Apache Struts was publicly disclosed.
On the 8th & 9th of March 2017, the Department of Homeland Security notified Equifax of the vulnerability and Equifax emailed their employees to instruct them that if they had Apache Struts running, to apply the patch within 48 hours.
Equifax's Automated Consumer Interview System (ACIS) failed to apply the patch.
On the 15th of March 2017, Equifax's security team scans the network for any system still affected by the Apache struts vulnerability, but the scan does not detect them because the device to monitor it is offline due to an expired certificate.
On the 13th of May 2017, attackers dropped web-shells (a web based backdoor) to gain remote access to the network.
Pre-activity
The attackers found unencrypted credentials that gave access to data outside of the ACIS environment.
Activity
Between the 13th of May and the 31st of July 2017, the attackers used the credentials to extract unencrypted personal data, including names, Social Security numbers, dates of birth, addresses, driver's license numbers, credit card numbers, and dispute documents from the databases of Equifax.
Equifax did not notice this, since the device used to monitor it was inactive for 19 months because it was not maintained. The device had an expired certificate.
Post-activity
On the 31st of July 2017 Equifax becomes aware of the breach and shut down the vulnerable system.
On the 7th of September 2017 Equifax informed the public of the breach.

Fig. 1. Crime script of Equifax data breach based on [13]

Figure 1 presents a crime script that describes the sequence of events that occurred during the Equifax breach. It provides the breach in the form of a script from the attackers' perspective. This is both helpful in analyzing what went wrong and was compromised, as well as in learning from the attack to improve security and ensure that it never happens again. Publishing such findings can help other companies avoid similar breaches in the future.

Figure 2 presents an attack tree that was derived from a crime script (see Figure 1), that shows the attack vector used by the attackers to attack Equifax. Unlike the crime script, which describes a specific sequence of actions, this figure provides a structured representation of possible attacks as nodes. Attack trees are generally

TScIT 43, July 4, 2025, Enschede, The Netherlands

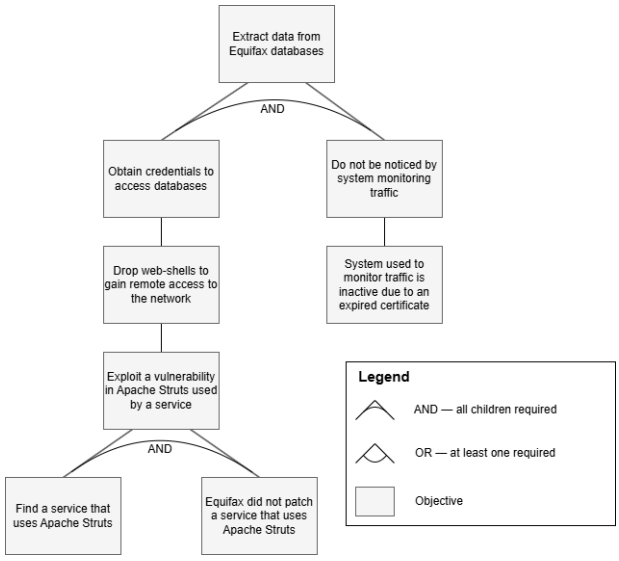

Fig. 2. Converting the crime script from Figure 1 into an attack tree.

non-exhaustive, as there are too many attack vectors (vulnerabilities) to breach a company. For example, in Figure 2, another path could be to phish an employee for data. It would simply be close to impossible to realize every possible vulnerability to attack the company and add them as nodes in the attack tree, especially because there are always developments in this area. Therefore, attack trees are generally scoped in a certain manner to ensure that the tree does not become too large and that the important attack vectors are included.

2 PROBLEM STATEMENT

Analyzing the result of the conversion of a crime script (Figure 1) into an attack tree in Figure 2 shows that an attack tree with only a single path is generated. This is as expected since an attack tree contains multiple attack vectors as paths, whereas a crime script explains a single attack vector in a sequence of events—a script. Because a path is a single sequence of events, the converted attack tree from a crime script would only have one branch/path.

The goal of this paper is to determine whether a detailed attack tree can be generated by replicating the crime script in a simulated environment. This attack tree would have to be more detailed and contain more paths, which are alternative paths that the attacker could have taken.

2.1 Research question

To achieve the goal of converting a crime script into a part of an attack tree, this paper aims to answer the following research question (RQ).

RQ: To what extent does recreating attack steps from crime scripts derived from cases such as the Equifax Data breach help improve the completeness of attack trees in terms of added paths?

The research question was divided into three sub-research questions, which, when answered, can be used to answer the research question.

RQ.1: Does hands-on vulnerability testing reveal steps missing from the crime script?

RQ.2: Could recreating attack steps from the crime script reveal hidden attack vectors that were not used in the original attack?

RQ.3: How much richer in nodes or more realistic does the attack tree become after incorporating the findings from recreating the attack?

3 RELATED WORK

Attack trees were popularized by Schneier in 1999 [17]. Attack trees "provide a formal, methodical way of describing the security of systems" [17]. They provide an overview of the attack vectors, which are represented as paths in the attack tree, that an attacker can use to break into a system. These hierarchical structures contain a root node which represents the attackers goal, and child nodes which represent ways to achieve that. They may also include metrics such as likelihood or cost; however, this is outside the scope of this paper. Since their introduction and adoption in the cybersecurity sector, several variations have been made, such as attack-fault trees, probabilistic attack trees, and attack-defense trees. Attack trees are useful for cybersecurity because they can be used to analyze which vulnerabilities should be prioritized for defense.

Crime scripts were popularized by Cornish in 1994 [6]. Crime scripts are a concept borrowed from cognitive science and criminology that describe the sequence of actions a criminal has taken in a specific criminal activity [6]. In cybersecurity, they have been adapted to model digital incidents and consist of actions taken before, during, and after a crime. [4] propose a structured and methodical process for populating such crime scripts using open source intelligence (OSINT), further formalizing their application. This provides, like attack trees, a look into the actions of a criminal and the steps they take to examine what security measures could be put into place to make such a crime less likely to happen.

Although crime scripts, sometimes referred to simply as scripts, and attack trees are well documented, they are often not employed together. There exists a gap in the existing literature on how these two can complement each other when the crime script is replicated through hands-on simulation.

Research on the Equifax data breach has focused on various aspects, such as its causes and impacts. According to [18], the breach was attributed to the failure to apply a patch to a vulnerability in Apache Struts, used by one of Equifax's public-facing software, which was known for months before the attack. Furthermore, [20] established weaknesses in Equifax's security practices, such as inadequate risk management and monitoring. Moreover, [18] emphasized the critical consequences for both the company and customers, highlighting the importance of timely application of security patches and competent security practices to mitigate such risks.

Hands-on simulations as a method to analyze attacks have been shown to give results in previous research. For example, [2, 5] demonstrate the value of modeling and simulating attacks to understand systemic vulnerabilities. Furthermore, [9] uses an automated

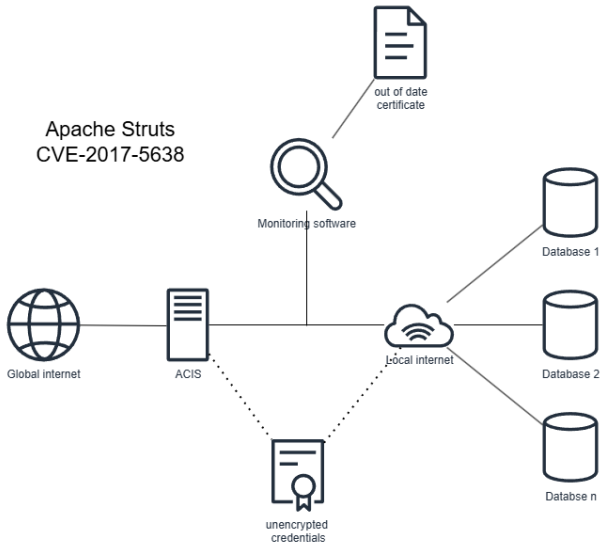

Fig. 3. Assumed relevant infrastructure of Equifax during breach.

testing framework to reconstruct attack graphs to correlate alerts with attacker actions, arguing that existing alert correlations are insufficient.

Despite this, no prior work has improved attack trees with crime scripts through a technical recreation of the steps outlined in the crime script. This paper directly addresses this gap by recreating the attack scenario from the Equifax breach. This allowed for a more detailed extraction of attacker actions and decisions that could be mapped onto an attack tree. In doing so, this research not only contributes to further analysis of the Equifax breach, but also provides an approach that can be used to analyze attacks in the cybersecurity sector.

4 METHODOLOGY

To answer the research questions, an isolated environment with the vulnerable software used by Equifax at the time of the breach was set up. The environment is in the form of a virtual machine. This environment contains an outdated version of Apache Struts that is vulnerable to the exploit (CVE-2017-5638) [1] used in the Equifax breach. Other databases were configured to contain dummy personal data, with credentials put in a configuration file in the folder of Apache Struts. Furthermore, since the device that was put in place to monitor traffic was not operational in the attack, it was omitted in the recreation but was taken into account during every step. The assumed infrastructure of the relevant system is shown in Figure 3.

First, the crime script was critically analyzed, then the (grey) literature was reviewed, and lastly the attack was reproduced. The steps outlined in the crime script were recreated in this environment to replicate the attack. During this simulation, the differences between the planned attack path of the crime script and the attack path taken in the simulation were noted. Furthermore, the other steps that the attacker could have taken to reach the goal were considered at each stage and were analyzed. Simulating and recreating attacks has been shown to yield valuable insight that traditional methods such as literature review or post-event analysis often overlook, especially in underexplored domains[2, 5, 9].

These findings were analyzed by determining whether they align with the attack in the crime script or represent alternative steps that could be added to enrich the attack tree. In addition, all findings were compiled into an attack tree to enrich it. This attack tree (see Figure 5) was then compared to the initial attack tree by metrics such as number of nodes, leaf nodes, unique paths to goal, average children per OR node and maximum tree depth and width to see if the goal of enhancing the attack tree was achieved. Afterwards, an enhanced crime script (see Figure 4) was created that also integrated all relevant findings to demonstrate what findings were found which can be compared to the initial crime script (see Figure 1).

4.1 (grey) Literature review

Due to a lack of technical depth in the papers on the Equifax breach, an alternative method was used to gather techniques to execute the attack. Furthermore, reputable sources, like the CVE database, where all exploits are indexed, are often the direct source of this sort of information in papers. Search engines like Google were used to search for reputable sources such as security labs that explained the attack in great technical depth. Only authoritative sources and/or reputable sources that are often used in papers, which were verified using Google Scholar, were used. Apart from authoritative sources such as the report from the U.S. House of Representatives Committee on Oversight and Government Reform which provided a report on the Equifax breach[13] and the CVE database maintained by the United States' Homeland Security Systems Engineering and Development Institute FFRDC, only information to execute the attack was gathered which was first verified during hands-on testing before being accepted. Furthermore, no formal models, proofs, or concepts were used from any source that was not a peer-reviewed paper. These papers were also evaluated by how reputable the journal they were published in was, how many citations they had, and if the content seemed of quality such as citations and proper academic terms were used.

First, the report on the Equifax breach[13] by the U.S. House of Representatives Committee on Oversight and Government Reform was analyzed for the sequence of events that lead to the attack. After verifying this information by checking if it aligned with what other papers showed, the information was used to create an initial crime script (see Figure 1) and an initial attack tree (see Figure 2) was created from this crime script.

Afterwards, the CVE database was inspected for requirements for the vulnerability, such as vulnerable software versions. Then, other sources were inspected for exact instructions on how to execute the attack, where sources that included verifiable reasoning, such as code[16], were prioritized. The common exploit[1, 16] which was verified during the hands-on testing, in which mostly the cmd value (command) differs, is as follows:

Content-Type: `${(#_='multipart/form-data').(#dm=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS).(#_memberAccess?(#_memberAccess=#dm):((#container=#context['com.opensymph`

```
ony.xwork2.ActionContext.container']).(#ognlUtil=#cont
ainer.getInstance(@com.opensymphony.xwork2.ognl.OgnlUt
il@class)).(#ognlUtil.getExcludedPackageNames().clear(
)).(#ognlUtil.getExcludedClasses().clear()).(#context.
setMemberAccess(#dm)))).(#cmd='whoami').(#iswin=(@java
.lang.System@getProperty('os.name').toLowerCase().cont
ains('win'))).(#cmds=(#iswin?{'cmd.exe','/c',#cmd}:{'/
bin/bash','-c',#cmd})).(#p=new java.lang.ProcessBuilde
r(#cmds)).(#p.redirectErrorStream(true)).(#process=#p.
start()).(#ros=(@org.apache.struts2.ServletActionConte
xt@getResponse().getOutputStream())).(@org.apache.comm
ons.io.IOUtils@copy(#process.getInputStream(),#ros)).(
#ros.flush())}
```

The payload is described in detail in Table 1.

4.2 Reproduction of the attack

To reproduce the attack, an environment was set up as shown in Section 4.3 (see Figure A.1 and Figure A.2). Findings discovered during the previous stage were verified. Steps from the initial crime script (see Figure 1) were recreated and findings were noted down. First, it was verified that with the payload shown in Table 1 the exploit worked (see Figure A.4 and Figure A.5). Then a webshell was made and uploaded via a Python script utilizing the payload (see Figure A.6). This payload was then used to search for database credentials and extract the prepared information from the database (see Figure A.7). The logs were also analyzed to determine whether the errors described by sources such as [16] were the cause of the exploit by referencing the Apache Struts source code (see Figure A.3). Between each step, including the setup, it was noted what access the attacker had at that stage and what other directions the attack could have taken.

4.3 Environment

Ubuntu 24.04.1, a Linux operating system (OS), was used in a virtual machine using VMware Workstation 17 Pro version 17.5.2. Here, a simple Apache Struts version 2.5.10 application was written in IntelliJ using a simple guide[8] and was deployed to recreate the attack. The Java Development Kit (JDK) used to compile the code was version 8u112, released in 2016. It was then executed using the server Java Runtime Environment (JRE) version 8u112 and deployed on Apache Tomcat 8.0.38, which was also released in 2016. A MySQL database version 24.04.1 was set up on the same machine, and the database credentials were stored in a configuration file in the Apache Tomcat files that hosted the Apache Struts application. Python scripts and curl commands were used to send requests to the server.

5 RESULTS

This section presents results from three sources: a critical analysis of the existing crime script for technical details, a review of the existing (grey) literature, and technical hands-on testing in a simulated environment. With this, our objective is to observe findings to answer the research questions, such as which steps are missing from the crime scripts, what hidden attack vectors there are, and eventually create an attack tree to analyze with the gathered information.

Table 1. Explanation of the OGNL code used for the exploit

OGNL code	Explanation
`${`	shows the evaluator that the following text is an (OGNL) expression.
`(#_='multipart/form-data')`	assigns the content-header to a random variable, since the content-type header needs to contain that otherwise the wrong path is taken in the code for our exploit.
`(#dm=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS).(#_memberAccess?(#_memberAccess=#dm):((#container=#context['com.opensymphony.xwork2.ActionContext.container']).(#ognlUtil=#container.getInstance(@com.opensymphony.xwork2.ognl.OgnlUtil@class)).(#ognlUtil.getExcludedPackageNames().clear()).(#ognlUtil.getExcludedClasses().clear()).(#context.setMemberAccess(#dm))))`	checks if the _memberAccess is on the value stack, and overwrites it. If it is not, it clears the blacklist the other way that was previously explained.
`(#cmd='whoami').(#iswin=(@java.lang.System@getProperty('os.name').toLowerCase().contains('win'))).(#cmds=(#iswin?{'cmd.exe','/c',#cmd}:{'/bin/bash','-c',#cmd})).(#p=new java.lang.ProcessBuilder(#cmds)).(#p.redirectErrorStream(true)).(#process=#p.start()).(#ros=(@org.apache.struts2.ServletActionContext@getResponse().getOutputStream())).(@org.apache.commons.io.IOUtils@copy(#process.getInputStream(),#ros)).(#ros.flush())`	checks if the machine is a windows or linux machine, executes the command in the variable “#cmd” which is “whoami” in this case, which returns the username of the machine, and adds this to the response to the request we made.

5.1 Insights from critical analysis of the crime script

Although Apache Struts most likely does not return its version when scanned on a network, it is easy to check whether the vulnerability works by modifying the header. An automated network scan can thus be run on the specific network of Equifax, or just the Internet in general, to find Apache Struts applications running with vulnerable versions.

An attacker would upload a webshell on the server, most likely in the form of a .jsp file, which would be dropped in the public facing directory and could thus be accessed on the website. This means that the traffic would be harder to detect and the webshell would

provide persistent access, even if the exploit in Apache Struts were to be patched. This webshell executes all GET and POST requests as shell commands and returns the output.

Unencrypted database credentials were found in a file[13]. It is unclear what type of file this was, but unencrypted database credentials are commonly placed in code, configuration files, or on a shared drive.

The data in the database were subsequently most likely extracted using the credentials through the webshell.

5.2 Findings from (grey) literature

One of the first requirements is to have an application running a version of Apache Struts 2 2.3.x before 2.3.32 or 2.5.x before 2.5.10.1 [14].

When a request is made whose content-type header does not match any expected value, an error is thrown, which includes the invalid content-type header as the message. An interceptor then looks for errors and calls the LocalizedTextUtil's findText method on it to retrieve the error message. Because the wrapped object containing the error does not have this message stored anywhere by the procedures the function takes, it will return the default message. However, this default message is first passed along to TextParseUtil's translateVariables method, which attempts to evaluate the message as an expression. Here, the unsanitized tainted user input is evaluated as an Object-Graph Navigation Language (OGNL) expression, which is an expression language, and opens up all sorts of possible exploits [16].

However, Struts has a blacklist for member access of classes and packages that causes OGNL not to be able to edit the response or execute any code. However, another oversight allowed a malicious actor to empty this blacklist and thus still be able to edit the response and execute arbitrary code. This is because while the blacklist also contained all the classes needed to access the blacklist, this blacklist only prevented from referencing a class member, and unfortunately a key referencing this object was already on the OGNL value stack. [16]

Furthermore, some versions of Struts even put the _memberAccess variable on the OGNL value stack, where it is as trivial as overwriting this variable with the default access one to circumvent the blacklist. [12]

Therefore, the common exploit[1, 16], in which mostly the cmd value (command) differs, is described in detail in Table 1.

5.3 Hands-on testing

During the setup phase, several observations were made from the choices that had to be made, and decisions that had to be considered more thoroughly. For example, while setting up the database containing the dummy sensitive data, a user had to be created that could access this database. This user was required to have a location from which it could be accessed. If set to "localhost", the user would only be able to access the database from the computer that hosts it. However, if it were set to "%", the user would be able to access the database from everywhere, given that they had the correct credentials. If set correctly, together with more categorized users with different access levels, it would have been less likely that an attacker could have accessed 48 unrelated databases[13].

With the environment fully operational, the Apache Struts vulnerability was successfully exploited using a payload that added a header to the response. The common payload mentioned earlier allowed for arbitrary code execution. It was first observed that the payload string containing the content-type header was extremely long, which could have been detected by a properly configured firewall[10] that would have dropped the request. Additionally, the payload caused the response to be malformed, adding the command output at the end of the headers. This caused some of the methods used to carry out this attack to fail and not print the response, complaining that the response was malformed. Furthermore, this could lead to detection by anomaly based[7] detection systems. Despite no action being defined for the multipart form-data header and the entirely incorrect content-type header, the response still returned a status code 200 due to the exploit that tampered with the response, which could aid in being undetected by monitoring tools that rely on error-based[15] alerts. Otherwise, without the payload, these two conditions would result in error codes.

With access to the console of the machine that hosts the web application, the attacker had access to almost everything on that machine. With a specially crafted payload, it was possible to upload a webshell as stated in the crime script (Figure 1), which provided persistent access even if the exploit were to be patched. With this webshell, commands could be performed, and the installation was searched for hard-coded, and/or any configuration files that contained credentials to a database. When such files were discovered, it was possible to print them with standard commands such as "cat" and view the credentials. With these credentials, gaining access to the database was trivial. With the ability to execute shell commands, a command-line SQL client was installed and connected to the database. From there, the prepared dummy data was found.

Another obstacle was determining how to extract the data from the database. Simply executing the command and outputting the result as text to the webshell would technically be possible but would not be very practical. This is because for that, especially with that much data, the body of HTTP traffic is not made and the formatting would also be lost. Exporting the result to a file and then downloading the file via the webshell proved to be the most efficient and covert method. With a misconfigured firewall that allows FTP and DNS traffic to the outside, it would also be possible to extract it using FTP and DNS Tunneling[19]; however, this would cause more unusual traffic that would be easier to notice.

Finally, everything from the initial research was verified during hands-on testing. For example, the attack also worked with the content-disposition and content-length header. Although other headers could have been used, the attackers may have preferred the content-type header due to wider availability of online documentation and examples. One thing that was overlooked in the crime script is that there is no mention of when the attackers first noticed that the server was vulnerable. There would have been a time interval between then and the crafting of a payload to upload the webshell.

5.4 Answering RQ.1: Steps missing from the crime script

The findings indicate that hands-on vulnerability testing does reveal steps missing from the crime script, albeit to a limited extent. An example can be seen in one of the findings, which outlines that there was no mention of when the attackers first noticed that the server was vulnerable. With these findings, an updated and enhanced crime script was created in Figure 4. This could provide valuable insight into the interval between identifying the vulnerability and exploiting it, indicating the time frame in which the system administrators have to patch the vulnerability after being informed of the unusual traffic that often accompanies vulnerability testing.

5.5 Answering RQ.2: Hidden attack vectors

The findings also indicate that recreating attack steps from the crime script can reveal hidden attack vectors that were not used by the attacker. For example, the attacker could have exploited the content-disposition header instead of the content-type header. However, it is more accurate to say that while there were other attack vectors, there were not exactly many hidden ones that surfaced due to the hands-on testing and could also have been inferred by critical analysis.

5.6 Answering RQ.3: Attack tree richness and realism

To assess how much richer in nodes or more realistic the attack tree becomes, an updated version was constructed using the findings and the results of the previous two sub-research questions (see Figure 5). Table 2 compares this attack tree with the initial attack tree in Figure 2, highlighting the increased number of nodes (8 vs. 47). It also has more alternative paths instead of just an attack tree with a single path. Moreover, an actual attack was carried out which was modeled by the tree, instead of being solely based on literature. Together, these two aspects make the attack tree more realistic.

5.7 Answering the Research question

With the sub-research questions answered, the main research question can now be addressed. Table 2 compares the initial attack tree in Figure 2 with the attack tree that incorporates the findings from recreating the attack steps from the crime script in Figure 5, highlighting the difference in the number of nodes, leaf nodes, paths, children per OR node, and maximum depth and width. It shows that the enhanced attack tree (see Figure 5) has more nodes (47 vs. 8) and more leaf nodes (28 vs. 3), demonstrating the increase in detail and granularity. It also shows an increase in the maximum tree depth (8 vs. 5) and width (12 vs. 2), illustrating the increased amount of detail in the attack paths. Furthermore, it shows that the enhanced attack tree has more unique paths to the goal (11664 vs. 1) and the increase in the choices the attacker had at each point was revealed by the increase in the average children per OR node (~2.21 vs. 1), which further demonstrates the inclusion of alternative steps for the attacker. Because the initial attack tree (see Figure 2) was based on the crime script with no modifications, it had a single path, as a crime script outlines a single attack. However, the attack tree in Figure 5 has many more paths. These structural characteristics provide formal evidence of the enhanced attack trees improved completeness, showing that recreating the steps in the crime script had a significant impact on the its completeness, especially in terms of added paths.

Preparation
A remote command injection (RCI) vulnerability found in Apache Struts was publicly disclosed.
The Department of Homeland Security notified Equifax of the vulnerability and Equifax emailed their employees to instruct them that if they had Apache Struts running, they should apply the patch within 48 hours.
Equifax's Automated Consumer Interview System (ACIS) failed to apply the patch.
Equifax's security team scans the network for any system still affected by the Apache struts vulnerability, but the scan does not detect them because the device to monitor it is offline due to an expired certificate.
Attackers create a payload that tests if a server is vulnerable to the exploit, and scan either the entire internet or target Equifax specifically.
Attackers notice that Equifax's ACIS system is vulnerable.
Attackers create or find a web-shell.
Attackers create a payload that uploads a web-shell and choose which header to exploit (Content-Type, Content-Disposition, or Content-Length).
Attackers dropped web-shells (a web-based backdoor) to gain remote access to the network.
Pre-activity
The attackers found unencrypted credentials, most likely in the form of a configuration file, hard coded on the website or on an open shared drive, which gave access to data outside of the ACIS environment.
The attackers found the database(s), most likely with the credentials, scanning the network or checking the running services.
Activity
The attackers used the credentials to extract unencrypted personal data, including names, Social Security numbers, dates of birth, addresses, driver's license numbers, credit card numbers and dispute documents from the Equifax databases, most likely through the web-shell, FTP or DNS tunneling.
Equifax did not notice this, since the device used to monitor it was inactive for 19 months because it was not maintained.
The device had an expired certificate.
Post-activity
Equifax becomes aware of the breach and shuts down the vulnerable system.
Equifax informed the public of the breach.

Fig. 4. Enhanced crime script incorporating new steps identified during critical analysis, literature review, and hands-on testing.

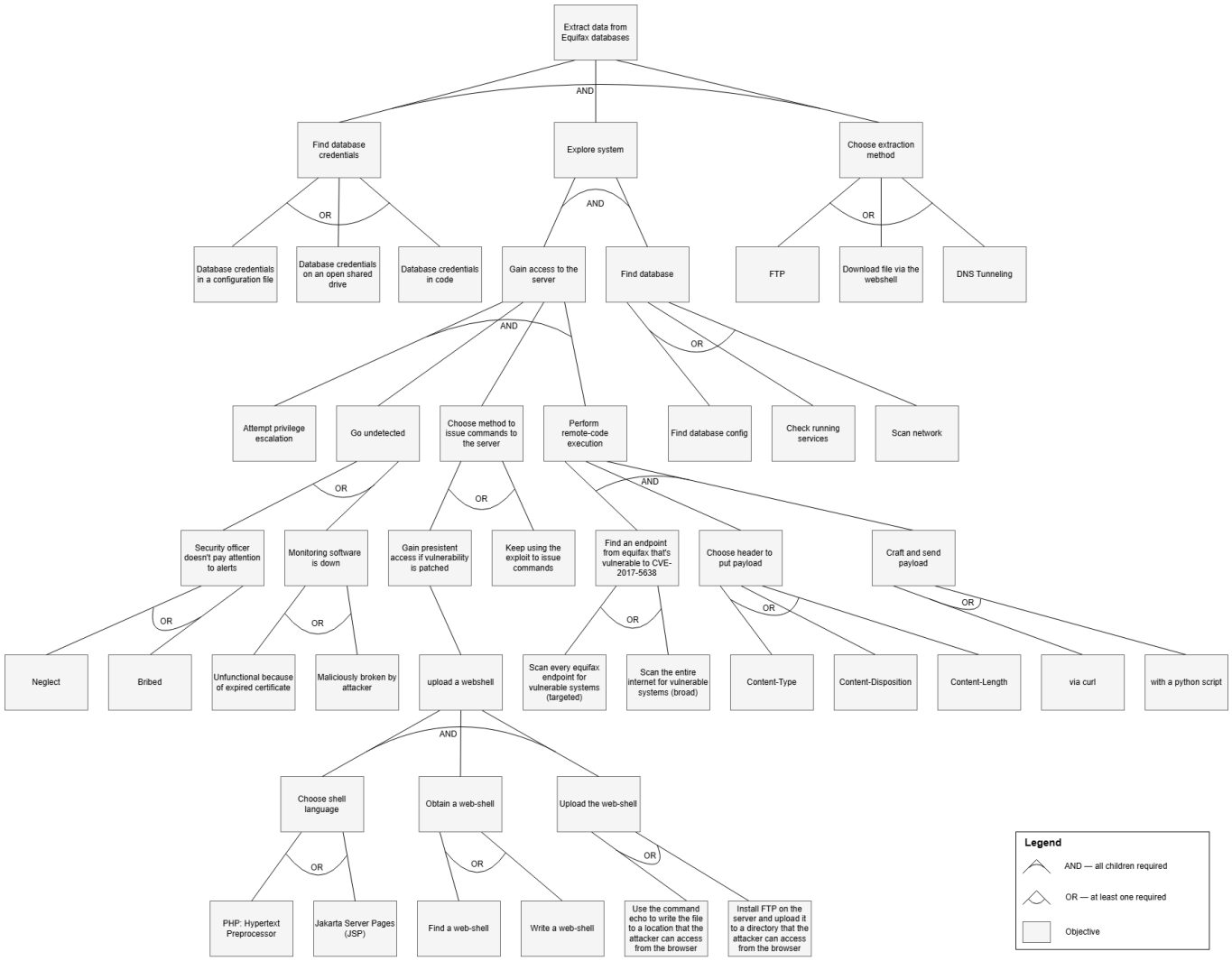

Fig. 5. Attack tree incorporating findings identified during critical analysis, literature review, and hands-on testing.

6 DISCUSSION

Despite the significance of the 2017 Equifax data breach, to the best of our knowledge, there is a notable lack of literature that reconstructs or analyzes the incident. Most papers that reference the breach either reference it in passing or focus on other aspects, such as its legal, political, and regulatory implications[3]. As a result, a technical breakdown of the attack and its possible variations remains underexplored in peer-reviewed research from reputable sources at the time of writing.

One paper by Zhang et al. [11] presents one of the few studies that attempts a technical reconstruction of the Equifax breach. Their work leverages a custom framework named Cybersafety, which highlights recurring security failures, such as the lack of a web application firewall (WAF) and exposed application versioning. However, their approach is based on existing literature, making assumptions where the exact details were unclear, and does not engage in formal modeling or recreation of the attack. Therefore, it fails to show what other steps the attacker could have taken, both in the assumed steps and where the steps were unclear or known.

This paper directly addresses this gap by recreating the attack scenario from the Equifax breach. This allowed for a more detailed extraction of attacker actions and decisions that could be mapped onto an attack tree. The approach goes beyond previous research by reconstructing the attack, resulting in observable and real-world findings rather than theoretical inferences. This resulted in not only a more detailed attack model, but also demonstrated how crime script analysis and attack tree construction can complement each other to uncover overlooked attack paths and refine existing ones.

Our research also shows that crime scripts can be enhanced by recreating the steps (see Figure 4). For example, the initial crime script lacked details on reconnaissance, such as first identifying the server as vulnerable. Furthermore, it also lacked details on how the

Table 2. Comparison of the attack trees in Figure 2 and Figure 5

Metric	Figure 2	Figure 5
Total number of nodes	8	47
Number of leaf nodes	3	28
Number of unique paths to goal	1	11664
Maximum tree depth	5	8
Maximum tree width	2	12
Avg. children per OR node	1	~2.21

data was extracted. Recreating the attack helped to expose these gaps.

In doing so, this work contributes more than just a refined model of the Equifax breach. It demonstrates the value of analyzing attacks by reconstructing the attack to enhance attack trees and crime scripts that can be used as a means to analyze both what went wrong and what else could have happened. For example, by identifying alternative payloads such as Content-Disposition instead of Content-Type and modeling different extraction types such as FTP or DNS tunneling, a significantly enhanced attack tree containing 47 nodes and over 11000 unique paths was created, compared to the single-path tree that was directly derived from the crime script.

Although the added depth to the attack tree and insight into the attacker's decision-making process make the process worthwhile, it should not be done for all cases. While the method is valuable, its time cost makes it more suitable for high-impact breaches than smaller, less significant ones, where standard approaches suffice. Thus, security officers can use this approach selectively, executing it only after a breach to better understand the possible attack vectors. Furthermore, this would be a valuable tool for specialized security teams and forensic analysts.

Nonetheless, the method has clear practical applications. Forensic analysts and incident response teams can use this method to operate in a preventive manner rather than only looking at what went wrong. This method with its alternative attack paths can help suggest additional protective measures to implement that future attackers cannot exploit should they uncover some vulnerability.

Screenshots illustrating the successful execution of the exploit and other key stages of the hands-on testing are provided in Appendix A.

6.1 Limitations

During our research, we initially used an older version of CentOS from 2016; however, problems such as expired SSL root certificates and removed repositories occurred. We later used Ubuntu, as the Linux kernel version did not have a significant impact on the attack.

Furthermore, other limitations include the fact that the exact software, other than Apache Struts, and software versions were not publicly available. In addition, the exact methods used by the attacker were not publicly disclosed. This led to some speculations, which were represented as different paths in the attack tree. Future research with fewer resource constraints could contact the company for more details, ensuring the realism of the tree.

Lastly, we note that these are our observations, which might impact the replicability of the results, as others performing this research might have other realizations. We emphasize that while the realizations might not be the same, the approach results in detailed findings that enhance the understanding of the attack.

7 CONCLUSION

This paper demonstrates the value of enhancing attack trees by recreating attacks from crime scripts. By simulating the steps taken by attackers, we identified other hidden attack vectors and gaps in the crime script, especially in the reconnaissance and data exfiltration phase, which would otherwise have been overlooked. Although this approach requires time and resources, it provides valuable insights into high-impact incidents. Ultimately, this method offers security professionals and forensic teams another framework to better analyze attacks.

This paper showed the feasibility; future work could attempt to create a more structured approach. Furthermore, future research could explore automating the process of conversing crime scripts into attack trees, integrating, for example, AI that can help analyze and produce findings.

8 ACKNOWLEDGMENTS

The author thanks Dr. Christina Kolb and Dr. Jan-Willem Bullée for their guidance and supervision during this research. The author also expresses their gratitude to Eren Kodal, Lilya Saliba and Farida Elsaadany for their continuous assistance and feedback.

REFERENCES

[1] Fred Bals. 2024. CVE-2017-5638: The Apache Struts vulnerability explained. https://www.blackduck.com/blog/cve-2017-5638-apache-struts-vulnerability-explained.html.

[2] A. Barreto, M. Hieb, and E. Yano. 2012. Developing a complex simulation environment for evaluating cyber attacks. In *Interservice/Industry Training, Simulation, and Education Conf. (I/ITSEC)*, Vol. 12248. 1–9.

[3] Hal Berghel. 2020. The Equifax Hack Revisited and Repurposed. *Computer* 53, 5 (2020), 85–90. https://doi.org/10.1109/MC.2020.2979525

[4] Spencer P. Chainey and Arantza Alonso Berbotto. 2021. A structured methodical process for populating a crime script of organized crime activity using OSINT. *Trends in Organized Crime* (2021). https://doi.org/10.1007/s12117-021-09428-9

[5] Fred Cohen. 1999. Simulating cyber attacks, defences, and consequences. *Computers Security* (1999). https://doi.org/10.1016/S0167-4048(99)80115-1

[6] Derek Cornish. 1994. THE PROCEDURAL ANALYSIS OF OFFENDING AND ITS RELEVANCE FOR SITUATIONAL PREVENTION. *Crime Prevention Studies* 3 (1994), 151–196. https://popcenter.asu.edu/sites/default/files/problems/stolen_goods/PDFs/Cornish1994.pdf

[7] D.E. Denning. 1987. An Intrusion-Detection Model. *IEEE Transactions on Software Engineering* SE-13, 2 (1987), 222–232. https://doi.org/10.1109/TSE.1987.232894

[8] The Apache Software Foundation. [n. d.]. How To Create A Struts 2 Application. https://struts.apache.org/getting-started/how-to-create-a-struts2-web-application. Accessed: 2025-05-12.

[9] Hao Hu, Jing Liu, Yuchen Zhang, Yuling Liu, Xiaoyu Xu, and Jinglei Tan. 2020. Attack scenario reconstruction approach using attack graph and alert data mining. *Journal of Information Security and Applications* (2020). https://doi.org/10.1016/j.jisa.2020.102522

[10] Imperva. 2020. Abnormally Long Header Line. https://docs.imperva.com/bundle/on-premises-knowledgebase-reference-guide/page/abnormally_long_header_line.htm.

[11] Ilya Kabanov and Stuart Madnick. 2021. Applying the Lessons from the Equifax Cybersecurity Incident to Build a Better Defense. *MISQE* 20 (2021). Issue 2. https://doi.org/10.17705/2msqe.00044

[12] Man Yue Mo. 2018. OGNL Apache Struts exploit: Weaponizing a sandbox bypass (CVE-2018-11776). https://securitylab.github.com/research/ognl-apache-struts-exploit-CVE-2018-11776/.

[13] U.S. House of Representatives Committee on Oversight and Government Reform. 2018. The Equifax Data Breach. https://oversight.house.gov/wp-content/uploads/2018/12/Equifax-Report.pdf.

[14] National Institute of Standards and Technology. 2017. NVD - CVE-2017-5638. https://nvd.nist.gov/vuln/detail/CVE-2017-5638.
[15] OWASP. [n. d.]. Logging Cheat Sheet. https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html. Accessed: 2025-06-17.
[16] Eric Rafaloff. 2013. An Analysis Of CVE-2017-5638. https://www.aon.com/en/insights/cyber-labs/an-analysis-of-cve-2017-5638.
[17] Bruce Schneier. 1999. Attack Trees. *Dr. Dobb's Journal* (1999). https://www.schneier.com/academic/archives/1999/12/attack_trees.html
[18] Jason Thomas. 2019. A Case Study Analysis of the Equifax Data Breach 1 A Case Study Analysis of the Equifax Data Breach. (2019). https://doi.org/10.13140/RG.2.2.16468.76161
[19] Faheem Ullah, Matthew Edwards, Rajiv Ramdhany, Ruzanna Chitchyan, M. Ali Babar, and Awais Rashid. 2018. Data exfiltration: A review of external attack vectors and countermeasures. *Journal of Network and Computer Applications* (2018). https://doi.org/10.1016/j.jnca.2017.10.016
[20] Ping Wang and Christopher Johnson. 2018. CYBERSECURITY INCIDENT HANDLING: A CASE STUDY OF THE EQUIFAX DATA BREACH. *Issues in Information Systems* (2018). https://doi.org/10.48009/3_iis_2018_150-159

A SCREENSHOTS OF THE HANDS-ON TESTING

This appendix contains screenshots illustrating the successful execution of the exploit and other key stages of the hands-on testing.

A.1 Screenshots of host

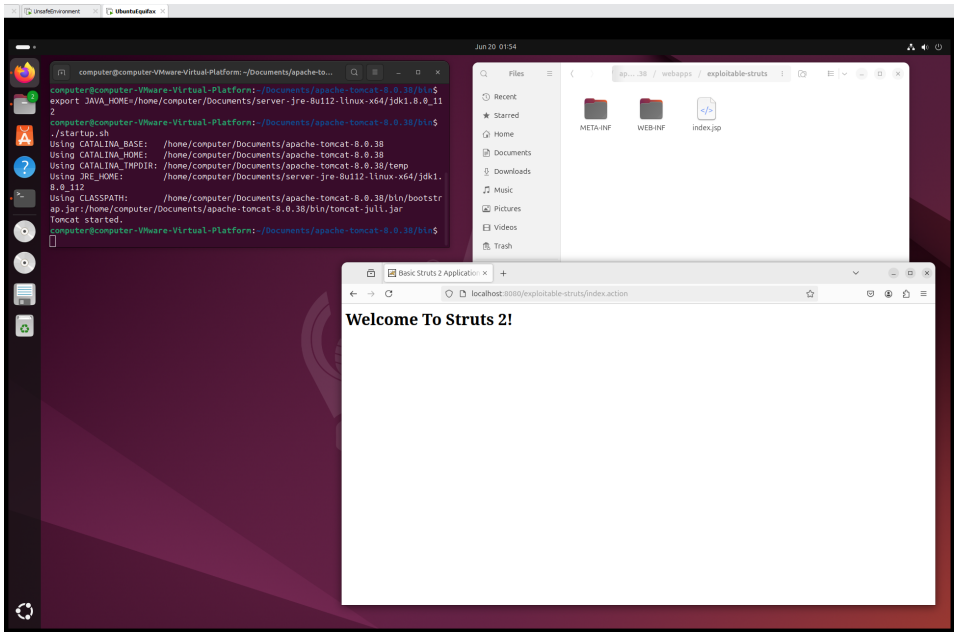

Fig. A.1. Screenshot showing tomcat starting on the host.

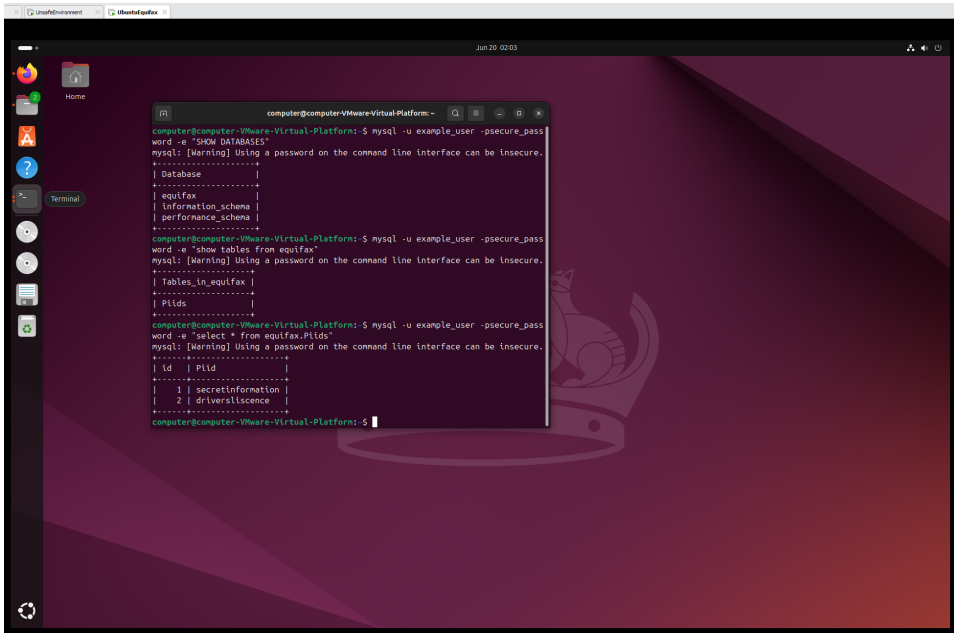

Fig. A.2. Screenshot showing data in the database on the host.

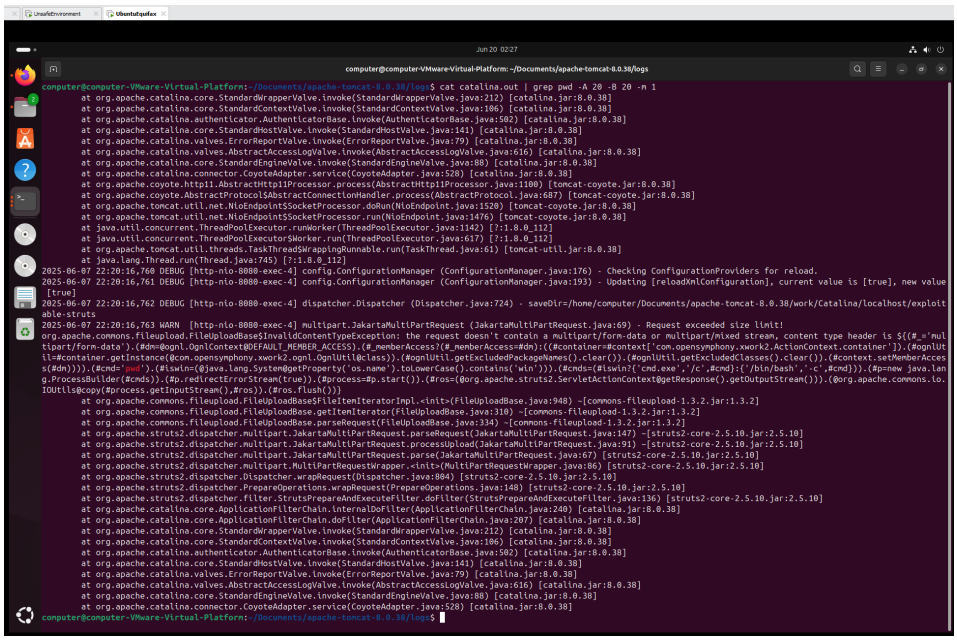

Fig. A.3. Screenshot showing log containing the printed payload in the error message.

A.2 Screenshots of attacker

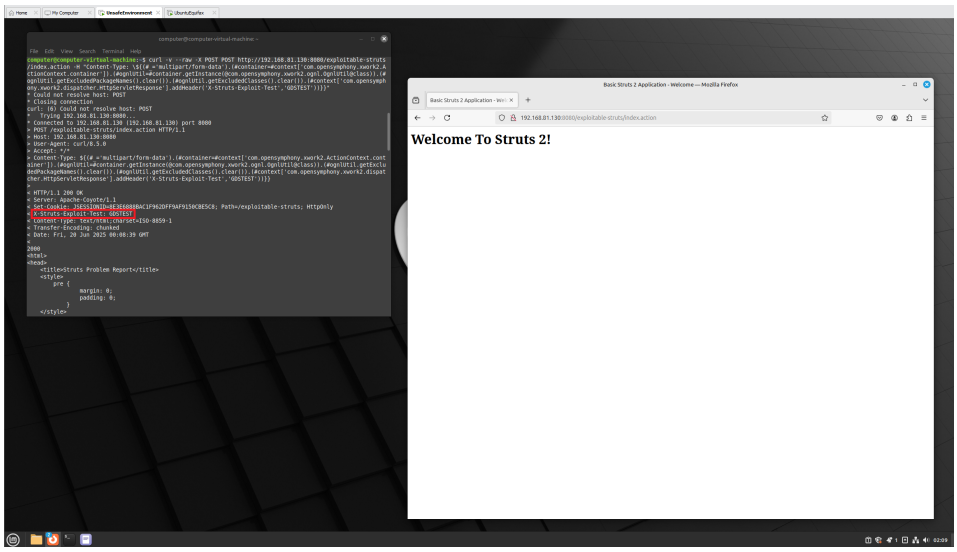

Fig. A.4. Screenshot showing server vulnerable by non-intrusive header injection method.

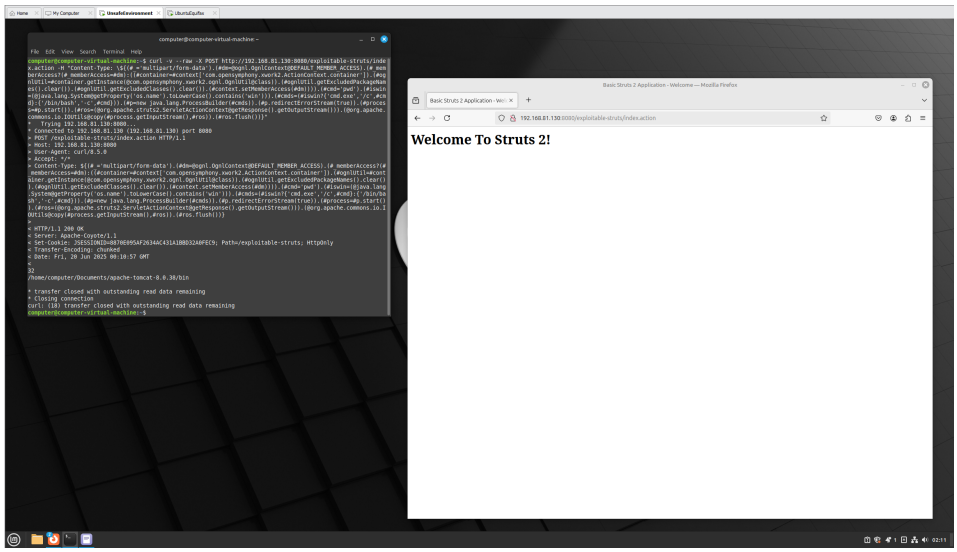

Fig. A.5. Screenshot showing command injection.

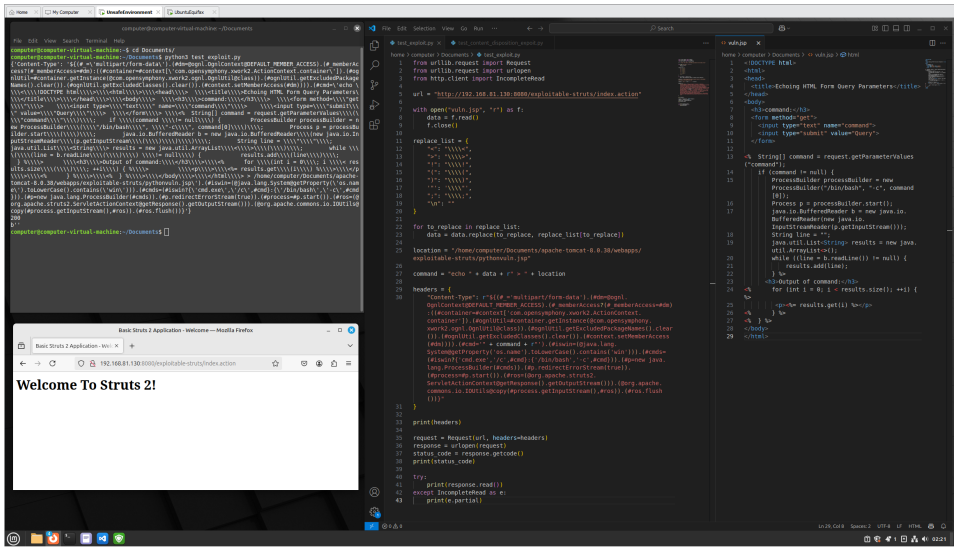

Fig. A.6. Screenshot showing process of uploading webshell. Includes python script and webshell

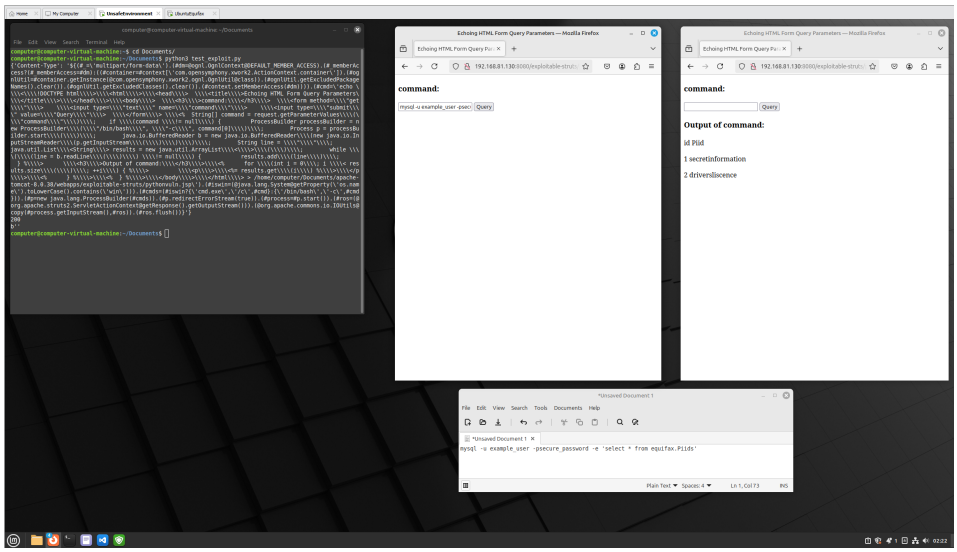

Fig. A.7. Screenshot showing process of exfiltrating data. Left browser shows initial page, right browser shows result after submitting command