A CIP-PSP funded pilot action
Grant agreement n°325188

Deliverable	**D1.4.1 Specification of Tool Group "Malicious or Vulnerable Websites"**
Work package	WP 1, Requirements & Specifications
Due date	M12
Submission date	31/01/2014
Revision	Revision 2
Status of revision	
Responsible partner	Fraunhofer FKIE
Contributors	Jan Gassen (Fraunhofer FKIE), Jonathan P. Chapman (Fraunhofer FKIE)
Project Number	CIP-ICT PSP-2012-6 / 325188
Project Acronym	ACDC
Project Title	Advanced Cyber Defence Centre
Start Date of Project	01/02/2013

Dissemination Level	
PU: Public	X
PP: Restricted to other programme participants (including the Commission)	
RE: Restricted to a group specified by the consortium (including the Commission)	
CO: Confidential, only for members of the consortium (including the Commission)	

Version history

Rev.	Date	Author	Notes
1	29/08/2013	Jan Gassen (Fraunhofer FKIE), Jonathan P. Chapman (Fraunhofer FKIE)	First draft
2	30/01/2014	Jan Gassen (Fraunhofer FKIE), Jonathan P. Chapman (Fraunhofer FKIE)	Incorporating feedback received from ACDC partners

Glossary

ACDC	Advanced Cyber Defence Centre
AHPS	Atos High Performance Security
CCH	Central Clearing House
SIEM	Security Information and Event Management
URL	Uniform Resource Locator
WWW	World Wide Web

Table of contents

Table of figures

1. Executive summary

In this document, we provide an overview to the analysis of malicious or vulnerable websites within the ACDC context. We define the requirements that tools contributed to the Malicious or Vulnerable Website Analysis must satisfy and describe the tools that were contributed so far. This document goes on to analyse the benefits and drawbacks of different communication models and defines a lightweight communication protocol for communications between a tool and the CCH.
Given the aforementioned analysis, tool-to-tool communication should be avoided in favour of immediately storing results as data elements in the CCH using the respective protocol. Tools that depend on such results may retrieve them through a subscription mechanism to provide additional analysis or data. Our solution supports the early dissemination of relevant information, including to the stakeholders, while reducing cost for implementation and maintenance.

2. Introduction

Since the world wide web not only gained its reputation as a key technology, providing efficient access to information, and even more so since the protocols associated with it turned out to be enablers for complex applications and business models such as cloud services, few sites with Internet access can afford to deny their users access to the WWW. This fact is exploited by malware authors, including botnet operators, by using the WWW for their malicious cause.
Malware authors use exploits for web browsers to infect new computers, web servers to distribute their malware and use or replicate the HTTP protocol for their botnet's command and control. Access to the WWW servers needed for these purposes can either be obtained by renting them from “no questions asked” or “bullet proof” Internet service providers or by gaining access to non-malicious servers, e.g. by exploiting vulnerabilities in their operating systems or applications running on them.
With no simple and reliable technique known for distinguishing malicious and non-malicious use of WWW technology, the abuse is likely to remain undiscovered until significant damage has been inflicted on a party. Thus, to provide comprehensive and effective protection against botnets or at least mitigation of their activities, the ACDC solution must provide tools to discover both the fact that a website is likely to be used for malicious activities and prevent the acquisition of servers for malicious activities by warning operators of insecure equipment or software about known vulnerabilities. With regard to websites, tools addressing these issues are organised in the ACDC tool group “Malicious or Vulnerable Websites”. This document introduces these tools, summarizes their requirements and defines how they will interact with each other and other components of the ACDC solution.
In section 3, we describe the relevance of website analysis in the ACDC context. Section 4 discusses the aspects that should be taken into consideration for website analysis, followed by a section on the requirements for tools that should be contributed to the Malicious or Vulnerable Website Tool Group. The tools that were contributed to the tool group are described in section 6. Section 7 goes on to describe the communication protocol for the tool group and finally, we summarise our conclusions in section 8.

3. Relevance of Website Analysis in the ACDC Context

Botnet operators have been abusing websites for malicious activities for many years. Thus, website analysis is an important part of the ACDC solution. Detecting this kind of abuse is challenging because attackers try to conceal their modifications as far as possible. The versatility of potential modifications further complicates a fast and reliable detection of compromised websites. As a consequence, detection should not only identify compromisation but also vulnerability of a website. By detecting and subsequently closing website vulnerabilities, website owners can prevent abuse in the first place. Therefore, website analysis within the ACDC solution covers both, the detection of malicious websites and the detection of vulnerabilities of websites. In this section, we briefly recapitulate the different ways for abusing websites with regard to botnet activities.

The most striking way in which websites are abused for botnet activities is by inserting malicious code that is used to perform attacks against visitors. Whenever a user visits a manipulated website with a vulnerable browser, its system can be infected with a malware to become part of a botnet. These so-called drive-by-download attacks may also be performed by websites that were created solely for this purpose. In order to attract more potential victims, these websites are often advertised, e.g. in spam email.

Another less obvious way to incorporate otherwise benign websites into botnet activities is to make the website itself, or the respective webserver, part of a botnet. Since webservers are commonly able to execute entire programs, e.g. scripts written in languages like PHP, these servers can be used to execute malicious programs that take part in botnet activities directly. To achieve this, botnets exploit website vulnerabilities that allow them to upload malicious scripts and have them executed by the webserver.

For both kinds of malicious website abuse, attackers can employ various techniques which makes it hard to apply a general approach for detecting all different kinds of website manipulations. Moreover, different information or even a different point of observation may be required to detect certain malicious websites. Websites carrying out attacks against their visitors may, for instance, be detected remotely, since attack-related code hast to be transmitted to the visitor. On the other hand, webservers participating in botnet activities directly cannot necessarily be identified by analysing the websites they deliver. Instead, network or device sensor data is required to detect malicious behaviour of the webserver.

The different types of information required for the detection of different kinds of malicious activities also directly affect the parties that are able to apply a particular analysis technique. Whereas drive-by-downloads can theoretically be detected by everyone, detecting webservers as part of a botnet may require direct access to the server or its network infrastructure. The latter techniques are therefore restricted to website owners or providers of respective infrastructures like hosting providers.

4. Detection and Mitigation Aspects Covered

Various aspects can be taken into account in order to identify malicious or vulnerable websites. Some of these aspects may provide certainty about the maliciousness of an examined website whereas others may only provide hints. Furthermore, the analysis of these aspects is commonly a trade-off between speed and scalability on the one hand and accuracy on the other. For the ACDC solution, analysis results for different aspects provided by different tools will be submitted to the central clearing house. This central storage of the results allows combining versatile information, aiding the reliability and effectiveness of malicious and vulnerable website detection.

The following sections outline the different kinds of information that can be used to identify vulnerable or malicious websites. They also discuss how each kind of information can be used to analyse websites, which analysis results can be gained and by whom the analysis can be applied. This section closes with a brief excurse on traffic redirection as a mitigation technique.

4.1. Malicious Website Analysis

Detecting malicious websites includes the detection of abused and otherwise benign websites as well as the detection of websites that were explicitly created to be malicious. Infecting otherwise benign websites can be achieved for example by exploiting cross-site scripting vulnerabilities. Thus, attackers may be able to include their own JavaScript or HTML

code on the attacked website. This code can either be used to directly attack the website's visitors or to redirect them to another malicious website.
Furthermore, it also covers the detection of successfully attacked webservers that directly take part in botnet activities like performing distributed denial-of-service attacks. Besides participating in a botnet as a client, an infected webserver could also be used as a C&C server to control the behaviour of other clients within a botnet. These attacks can be performed for example by uploading a malicious script to the server and using a file inclusion vulnerability to execute it.

4.1.1. *Delivered content*

Webservers deliver heterogeneous content like HTML documents, PDF documents or Flash content to a client's browser. The browser subsequently interprets the received content itself or by providing it to the respective plugin or external application. To exploit vulnerabilities in the client's interpreting application, the malicious code has to be transmitted to the client. Thus, by analysing the received content, such attacks can be detected. Since attackers are aware of this possibility, they try to obfuscate the attacks as far as possible, making it considerably harder for adversaries to detect them. Furthermore, attacks may only be carried out against victims from certain geographic locations or take the HTTP referrer into consideration. As a result, the content delivered by webservers can be used to detect attacks against clients, e.g. drive-by downloads, but this can be challenging in practice.

4.1.2. *Website source code*

Apart from static HTML documents, many websites rely on dynamically generated content. This content can be generated for example by an individual set of scripts or by complete content management systems. These scripts usually generate HTML documents on demand that are then transmitted to the client. The code generating the page on the server remains invisible to a website's visitor. In order to detect malicious scripts uploaded to the server, which are used to turn the webserver into a botnet client, the website's source code as well as uploaded files can be scanned for malicious content. Similar to other detection techniques, malicious content may be heavily obfuscated, rendering a reliable detection hard. Furthermore, direct file access to the webserver is required to analyse a website's source code.

4.1.3. *Network sensor data*

Observing the communication of a webserver with other entities over the network can provide clues about whether a webserver is part of a botnet in two ways. First, the bot script may either actively contact the botnet's C&C server or be contacted by botnet clients. Secondly, the bot script may take part in botnet activities like sending spam or launching DDoS attacks. Both activities require network communication and can thus be detected by analysing network sensor data. This however requires the sensor to be in a position where the entire communication of a webserver can be observed. Furthermore, especially command and control messages may be difficult to differentiate from regular traffic.

4.1.4. *Device sensor data*

Many attacks against webservers start with sending specially crafted HTTP requests to the webserver. These requests can be used for example to exploit XSS or SQL injection vulnerabilities. When such a request is send to a webserver, it can be logged for further analysis. This kind of logging for example is enabled by default for Apache webservers.

By deploying a device sensor analysing a webserver’s log files such attacks can be detected, indicating an infection. Furthermore, other indicators can be observed on webservers as on any other computer system, e.g. the creation of suspicious processes. Accessing this data for analysis however requires direct access to the webserver and since there is no precise a priori knowledge on how to identify an attack with respect to the indicators monitored by such a solution, it may neither be able to detect each possible attack nor be able to completely avoid false alarms.

4.1.5. *Reputation data*

Besides observing a website or a webserver directly, other information can be used to indicate a malicious website as well. One kind of information that can be exploited to achieve this is reputational information. If a website has been abused to deliver malware repeatedly in recent past, it is likely that this website will deliver malware again. As another example, if a website is frequently mentioned within spam emails, this may indicate malicious activities related to that website. To apply these techniques, no special permissions are required to access the webserver. However, such data can only indicate whether a website is malicious, i.e. it does not provide certainty.

4.2. *Vulnerable Website Analysis*

If a malicious activity is detected for a website which is known to be legitimate, this is a strong indicator for a vulnerability existing in the website’s software that has been successfully exploited. This kind of detection however requires that the website is already infected, which could have been prevented if the respective vulnerability had been detected and closed in time. Consequentially, it is also important to analyse websites for vulnerabilities that have not yet been exploited in order to prevent websites from being infected.
The respective approach is similar to the analysis of malicious websites even though it has a different goal. Therefore, some aspects of a website that would be analysed are similar to those used for malicious website analysis. These aspects as well as how they can be used to identify vulnerable websites are described in the following section.

4.2.1. *Delivered Content*

Even though the content delivered by webservers does generally not contain information about vulnerabilities, i.e. the vulnerability is not present within the content itself, it can still be used to indirectly identify potential vulnerabilities. In many cases, the delivered content contains information about the generating content management system, for example within a Meta-tag. If there are CVEs for the given version of that content management system, the website is very likely to be vulnerable. This technique does not require direct access to the analysed webserver.

4.2.2. *Website Source Code*

Instead of using the content generated by a CMS, its source code can be scanned for vulnerabilities directly. This can be done using signatures for known vulnerabilities, especially for common CMSs. Furthermore, website scripts can be scanned for vulnerable API calls, providing hints about a website’s potential vulnerabilities. Logic errors or other programming flaws that may lead to vulnerabilities are difficult to detect in general and thus, these techniques may not be able to reliably detect a website’s vulnerabilities. Furthermore, these techniques require direct access to the webserver and can thus only be applied by website owners or their service providers.

4.2.3. *Vulnerability Scanner*

Vulnerabilities of a website can also be detected by active probing, similar to what is done by attackers. To do this, special requests are sent to a website and its reactions are observed. This technique can be used to e.g. detect XSS or SQL injection vulnerabilities by automated trial and error. Since these requests are however similar to real attacks, applying this technique is usually not legal without explicit permission from the website's owner. Thus, while this technique can in principle be used by anyone, it may be prohibited altogether or limited to people or organisations acting on a direct mandate from a website's owner.

4.3. *Redirection of Malicious Traffic*

Detecting vulnerabilities in websites is feasible for detecting vulnerabilities known in advance only. Active vulnerability scanners on the other hand may be able to detect certain unknown logic errors, or other programming flaws that might lead to vulnerabilities, but are restricted in their application to only a few cases. In case these two approaches are not applicable, either due to technical or legal restrictions, there is another way of preventing a website from being infected without actually knowing about particular vulnerabilities. This can be done if a particular type of attack is known, even though it is unknown whether this attack can be applied to a certain website. SQL injection attacks for example commonly feature a characteristic pattern within POST or GET variables that are sent to a webserver. If this pattern is detected within a request, the request can be blocked in order to protect a potentially vulnerable website. As a result, an attacker cannot proceed even though the vulnerability she wanted to exploit was not known in advance.
This approach can be taken even further by redirecting suspicious requests to a honeypot system, offering the opportunity to analyse the attack in detail. This analysis may then result in the discovery of actual vulnerabilities, which can improve the security of the real website.
The functionality required for redirecting malicious traffic to a dedicated analysis server can be implemented on various levels. In the following section, these different levels are briefly described.

4.3.1. *CMS Plugin*

If a CMS features a central component for processing request, this component may allow filtering or redirecting requests found to be malicious. If a given CMS features a plugin architecture, it may be possible to implement such an approach without modifying the actual CMS. This approach can be applied by website owners without requiring changes to the webserver or its operating system. On the other hand, the central request-processing component may be vulnerable itself and thus not be able to prevent such attacks.

4.3.2. *Webserver Plugin*

Requests sent to a website are processed by the webserver first, for example to determine which file of a website was requested by the client. Popular webservers allow for adding plugins that interact with the webserver and may change the way a request is processed. By using such a plugin to scan requests for malicious patterns, requests can be redirected before they reach a potentially vulnerable website. This approach further allows to protect all websites that are hosted by one webserver at once. On the other hand, it does require modifications of the webserver and thus the respective level of permissions.

4.3.3. *External tool*

Malicious requests to a website may also be detected by using an external tool, e.g. analysers similar to an IDS. Such a system could scan for malicious patterns on network packet level and redirect such packets to another host. This approach would prevent malicious requests from reaching the targeted webserver in the first place and would thus also protect all hosted websites. On the other hand, this approach is more challenging since detecting malicious request on network packet level is significantly more difficult than on application level. Furthermore, this approach would not be able to deal with encrypted requests and would thus require permissions to install new software on the webserver's host to be able to process such requests.

5. General Requirements

In this section, we describe the general requirements for solutions that should be contributed to the ACDC project as a part of the Malicious or Vulnerable Website Analysis Tool Group. Unless stated otherwise, each of these requirements has to be fulfilled by any such tool. Additional requirements may be defined for individual tools.
In the following, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.
Where the use may not be immediately clear from the context, we use OR to indicate a non-exclusive or. Thus, α OR β implies that either the option α, the option β or both options α and β are valid at the same time.

5.1. Participation in the Tool Group

To participate in the Malicious or Vulnerable Website Analysis Tool Group, a tool MUST provide data OR analysis that supports the identification of websites serving malicious content OR vulnerable websites.

5.2. Communication Interface

Tools participating in the Malicious or Vulnerable Website Analysis Tool Group MUST implement the communication interface described in this document. Any tool in the tool group MUST react in the described manner, even if a specific element of the interface specification is not applicable for it.
Tools MAY implement additional interfaces for interacting with specific tools OR third-party applications where such interaction is not feasible using the interface defined in this document. Further details are defined in the Input and Output sections.

5.3. Input

Tools participating in the Malicious or Vulnerable Website Analysis Tool Group MAY acquire input from appropriate sources directly, require input from other tools or third-party sources or use any combination of these methods to provide their functionality. Tools SHOULD use the communication interface described in this document for acquiring data from other tools but MAY use other means where the interface is not appropriate. This is the case in particular if the data is unprocessed, of large volume and unlikely to be exploitable by other tools in the solution, e.g. raw network packet dumps or unfiltered log files.
Further details MAY be specified for individual tools.

5.4. Output

Tools participating in the Malicious or Vulnerable Website Analysis Tool Group MUST provide their analysis results OR data to the CCH, using the communication interface described in this

document. They MAY implement additional interfaces, with respect to the reasoning given in section 5.3.
Data transmitted by tools MUST provide a reasonable level of abstraction with regard to the subject or events analysed. Tools MUST NOT transmit large volume raw data such as PCAP packet traces or log files but MAY submit an indicator for their availability instead.
Tools in the Malicious or Vulnerable Website Analysis Tool Group MUST only provide data to the CCH that is associated with a particular website OR server providing a website or websites. When submitting a respective data set, it MUST include sufficient details to allow associating it with the respective website or server.
The output MUST provide hints about the maliciousness OR vulnerability of the associated URL or server. This can be either specified directly, i.e. specifying whether a website is malicious or vulnerable, or indirectly by providing data that can be processed by other tools to allow for or improve reliability of such a classification.
Each tool SHOULD provide reliable data or information. If a tool cannot guarantee that the data or information provided is correct, it should indicate the degree of certainty with regard to the respective statement. Certainty SHOULD be based on objective measures, e.g. the precision achieved in ground truth experiments, but MAY be based on less objective measures such as educated guesses, if obtaining an objective measure is not feasible for a given tool.
When a tool generates additional data or information regarding a data set already stored in the CCH, it MUST provide the unique ID of that data set in the submission of its result to allow for that data set to be updated. If a tool aggregates or provides analysis based on several data sets provided by the CCH, it SHOULD include their IDs in the data set submitted to the CCH.
Further details MAY be specified for individual tools.

5.5. Documentation

Tools may be provided as a service, a solution or an appliance. A service is maintained and operated by and only by the tool provider, receiving data from and submitting data to the CCH OR other parts of the solution. A solution is a programme or a set of programmes that is maintained by the tool provider but should be deployed by partners (possibly including the tool provider) or third parties. An appliance is a machine (explicitly including virtual machines) that will be pre-configured by the tool provider but deployed on a partner’s or third party’s premises.

5.5.1. General Requirements

A tool’s documentation MUST fulfil the requirements of all the deployment models (service, solution, appliance) that will be used for that tool within the ACDC context. The documentation SHOULD explain the data or information provided to the CCH with a reasonable level of detail, allowing for them to be leveraged both for additional services/solutions by ACDC partners as well as end users.
There is no required data format for the documentation. Partners SHOULD use non-proprietary, platform-independent data formats which are easy to maintain for the tool provider. Where such a format is not being used, the documentation MUST be provided both in the original form as well as in a format that may not be adequate for editing but is both non-proprietary and platform-independent (e.g. plain text, HTML or PDF). It MUST be made available to all ACDC partners through an appropriate channel.

5.5.2. Service

The documentation for a tool providing a service to the ACDC solution MUST state the contact details for both the person or department responsible for developing the tool (in particular for bug reporting and/or feature requests) and for ensuring the availability of the service. If availability of the service depends on the availability of a third party service, that service and the person, company or organisation operating that service MUST be stated in the documentation and specific contact details SHOULD be stated, where available. Tool providers SHOULD provide this information for any third party service they rely on, even if the availability of their own service does not depend on it.

5.5.3. Solution

When a tool is provided as a solution to be operated by ACDC partners, the documentation MUST include the contact details for the person or department responsible for developing the tool (in particular for bug reporting and/or feature requests). Where the person or department providing support for deploying a solution are not the same as for its development, the documentation MUST include their contact details as well.
If a tool relies on third party components OR services, they MUST be listed in the documentation, providing both information on how to obtain the component/access to the service and any licenses OR service plan required as well as how to obtain support. For commercial components and services, this SHOULD indicate the suggested licensing/service plan and costs.
The documentation SHOULD describe all necessary steps for deployment and MUST state any specific requirements regarding its operating environment, e.g. which operating systems it is known to be compatible with or what libraries must be installed in that environment.

5.5.4. Appliance

The documentation of a tool provided as an appliance MUST provide the contact details for the person or department responsible for operating/maintaining the given appliance. If the appliance requires access to external input sources, they MUST be provided in the documentation along with information on how to obtain access to them. If the appliance processes data or information that is not actively supplied specifically to it by the operator, i.e. it acts as a sensor, the documentation MUST describe what data OR information is acquired by the appliance and to what level of detail it will be available in normal operations.
Any method that allows the tool provider to gain access to the appliance without requiring immediate interaction with the partner hosting the appliance (e.g. through an SSH or RDP server available over a public network) MUST be listed in the documentation. For each of those methods, the tool provider MUST describe how the respective interface is protected against misuse.

6. Tools in Tool Group 1.1.4

This section gives a brief overview to the tools contributed to the ACDC Malicious or Vulnerable Website Analysis Tool Group. While it does provide a brief description of their inputs and outputs, further details will be provided in their documentation in accordance with the requirements defined in section 5.5.

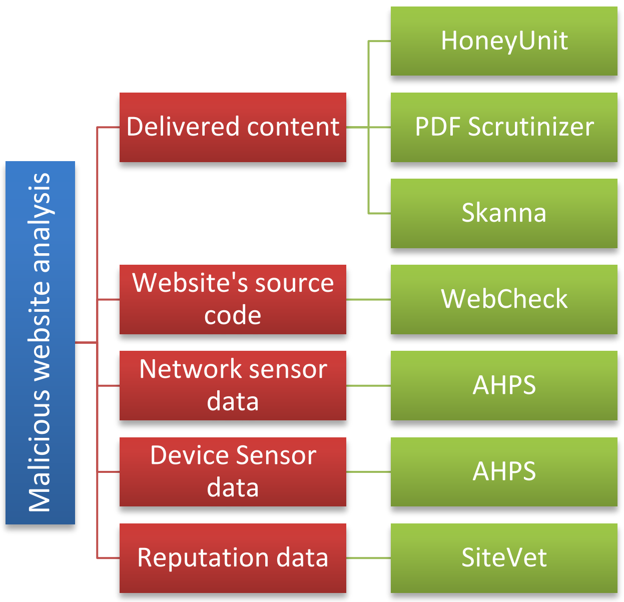

Figure 1 : Covered aspects by tool for malicious website analysis

Figure 1 shows all tools of tool group 1.1.4 that are used for malicious website analysis. Each tool is assigned to its respective aspect of malicious website analysis. If a tool covers more than one aspect, it is assigned multiple times respectively. As indicated by the figure, most tools focus on the content delivered by webservers since this information is publicly available. Note that all detection aspects with respect to malicious website analysis are covered by the current set of tools, allowing tool group 1.1.4 to provide comprehensive information about malicious websites.

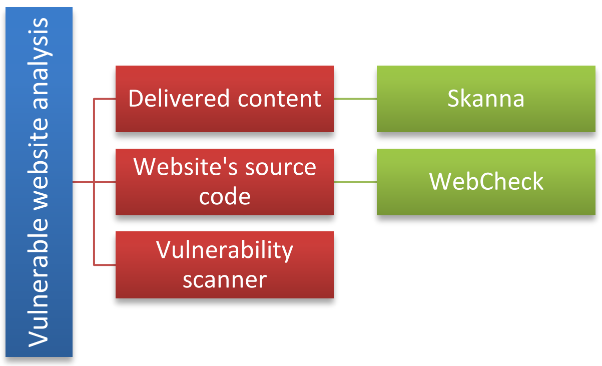

Figure 2: Covered aspects by tool for vulnerable website analysis

The tools provided for vulnerable website analysis are displayed in Figure 2. Respective tools are again associated with the aspect they cover in their analysis. As shown, two tools from tool group 1.1.4 explicitly focus on the detection of vulnerabilities in addition to malicious website analysis. The current set of tools does not include active vulnerability scanners that probe a website for potential vulnerabilities. It is however questionable whether such tools can be used within the ACDC solution given the legal restrictions discussed above.

Tools requiring identifier	Tools requiring lower-level data
• HoneyUnit	• AHPS
• PDF Scrutinizer	• SiteVet
• Skanna	
• WebCheck	

Figure 3: Required information by tool

Figure 3 outlines the different types of information required by the current set of tools. Most tools within tool group 1.1.4 use incoming information as an identifier to determine parameters for their analysis. These tools either use URLs or domain information to identify the website to analyse or need signatures that are used to scan a particular set of URLs. Tools that require lower-level data on the other hand augment submitted data with reputation information or analyse the data for particular attacks.

As described above, the current set of tools from tool group 1.1.4 covers almost all aspects of malicious or vulnerable website analysis. This allows providing versatile information to the central clearing house and thus to different stakeholders. To provide this information, tool group 1.1.4 also requires heterogeneous data for analysis, as described for the individual tools below.

6.1. AHPS

AHPS (Atos High Performance Security) service is a special SIEM (Security Information and Event Management) service provided by Atos. SIEM technology enables real-time analysis of security events generated by network devices, servers or applications. Event data is combined with contextual information about users, data and assets. AHPS deals with real-time monitoring, correlation of events, notifications, reports and console views.

In the context of malicious or vulnerable website analysis, AHPS does not scan or analyse websites directly. Instead, the system focuses on the infrastructure that hosts and supports websites. AHPS would be able to detect anomalous behaviour to identify a potential threat or detect that a server is under attack. Depending on the kind of attack observed, this may indicate a website being manipulated in a malicious way. The information generated by AHPS can then be used as input to other ACDC tools and stored in the CCH for further analysis. Other ACDC tools may use this information to analyse attacked websites, e.g. to determine whether these websites started delivering malicious content.

AHPS could further provide the ability to cross-reference event data signatures with vulnerability scanner data by means of an optional service. Using this optional service, notifications (encapsulated by AHPS events) are automatically generated when an attack is attempting to exploit a vulnerable system. This can be accomplished through an exploitation detection feature and the connection to other components such as intrusion detection and prevention systems as well as enterprise vulnerability scan results. AHPS has capabilities to provide cross-references between event data signatures and vulnerability scanner data. This can be used to generate feeds containing information about vulnerabilities and threats, normalized event signatures and associated remediation information.

6.1.1. Input

To observe website supporting infrastructures, AHPS is able to gather input from different event sources:

- Security perimeter: Devices and software used to create a security parameter for the observed environment.
- Operating systems: Events from different operating systems running in the network.

- Referential IT sources: The software used to maintain and track assets, patches, configurations, and vulnerabilities.
- Application events: Events generated from the applications installed in the network.
- User access control: Events generated from applications or devices that allow users access to company resources.

For that purpose, AHPS uses a set of predefined connectors:

- Audit Connector
- Check Point LEA Process Connector
- Database Connector
- Data Generator Connector
- File Connector
- Process Connector
- Syslog Connector
- SNMP Connector
- SDEE Connector
- Sentinel Link Connector
- WMS Connector

The AHPS takes the input from these connectors and converts the data into a textual map that can be processed by the collectors. Collectors parse and normalize this map and create an AHPS Event, categorizing it according to the AHPS taxonomy of events. The AHPS Event is enriched with additional source-specific data and, depending on the collector, additional contextual metadata such as identity, host, vulnerability, or custom mapped metadata. AHPS Events are later processed by a real time display, correlation engine, dashboards, as well as by the AHPS backend server.

6.1.2. Output

AHPS provides information to the CCH in form of AHPS Events. These events are formatted based on the Distributed Audit Services (XDAS) standard.
The AHPS Event model defines three separate actors in any event record connected by an action:

- The Initiator causes the event to occur by taking action against the Target
- The Target is the resource affected by the Action
- The Observer detects the Action taking place and generates an event record to describe it.

For each of them, AHPS events contain, amongst others, information about the domain, host name, IP address, service name, service port, device name, asset criticality, etc. By cross-referencing event data signatures and vulnerability scanner data, events may also contain attack names.

6.2. HoneyUnit

HoneyUnit, provided by Fraunhofer FKIE, is a generic security tool to analyse the runtime behaviour of websites and SVG documents. HoneyUnit is not a client honeypot by itself, but a test framework designed to provide extensive information on the runtime behaviour of websites examined. This information can be used by unit tests in order to apply heuristics or to check for particular exploits. Since the combination of such unit tests and HoneyUnit will create a client honeypot, HoneyUnit can be referred to as a generic analysis component for client honeypots.
To simulate the runtime behaviour of examined web pages, HoneyUnit is based on HtmlUnit, which allows modelling HTML documents. It is able to simulate different browsers and can be extended by custom JavaScript elements. Furthermore, HtmlUnit allows accessing HTML

elements programmatically, which allows creating unit tests for example to check for particular content of an examined web page. To execute embedded JavaScript, HtmlUnit uses Mozilla Rhino, which is an open source JavaScript engine written in Java. HoneyUnit is an extension for HtmlUnit, providing the framework with the capabilities required for examining the runtime behaviour of executed JavaScript. To provide a higher level of authenticity during analysis, the HoneyUnit framework is able to perform regular user interactions as hovering or clicking displayed elements. An overview, on how the analysis is performed by HoneyUnit is shown in Figure 4.

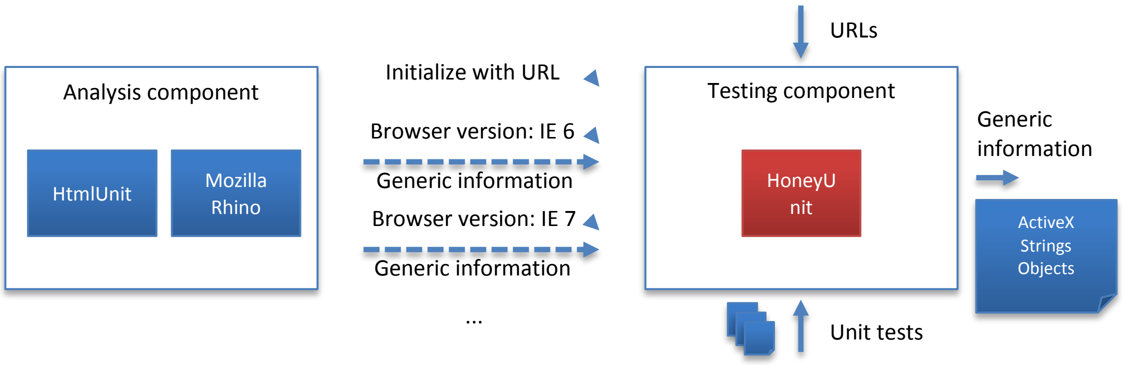

Figure 4: Overview about the HoneyUnit analysis process

Fraunhofer FKIE provides a set of HoneyUnit tests for detecting malicious websites. Each of those tests checks for various malicious characteristics. Through these tests, examined websites are checked for various suspicious structural and behavioural properties:

- Hidden iFrames: Cross-site scripting commonly relies on hidden iFrames to inject malicious content into otherwise benign websites.
- Obfuscation: Since malicious JavaScript code commonly is highly obfuscated, obfuscation is used as another indicator for malicious websites.
- Shellcode: HoneyUnit is able to detect shellcode used by malicious websites by applying dynamic analysis techniques.
- Suspicious variables: Strings or arrays that are passed as arguments to JavaScript functions are checked for suspicious patterns, e.g. *".*cmd.exe .*"*, which may be used to launch the windows command line interpreter.
- Heap spraying: Many malicious websites utilize heap spraying techniques to exploit vulnerabilities in a web browser. This requires allocating large strings or arrays containing repetitive content, which is detected by a respective test.
- Signatures: A signature-based detection of known exploits is applied on JavaScript methods and parameters during runtime, making it resistant against common obfuscation techniques. Therefore, this test can be used to reliably detect known exploits after having created the respective signature.

6.2.1. Input

HoneyUnit in conjunction with the tests provided is designed to analyse websites directly, i.e. by analysing the content that is delivered by a webserver. Therefore, HoneyUnit has to be provided with an URL, pointing to the website that should be analysed. In addition, HoneyUnit can also be used to analyse downloaded HMTL files. Therefore, HoneyUnit can also be provided with suspicious HTML content directly.

HoneyUnit allows websites to be scanned for known vulnerabilities by using custom signatures. Therefore, additional signatures could also be provided to HoneyUnit in order to identify current CVEs. Signatures to detect vulnerable ActiveX method calls contain the following information:

- Name: A short description of the signature

- CVE: The respective CVE, e.g. CVE-2008-0090
- Class id: The id of the according ActiveX object, e.g. D050D736-2D21-4723-AD58-5B541FFB6C11
- Method name: Name of the vulnerable method, e.g. SetPassword
- Argument: Argument value required to exploit the vulnerability, e.g. required argument length or contained patterns.

6.2.2. Output

When its analysis is complete, HtmlUnit displays the list of tests that indicated malicious or suspicious behaviour. For each test, a short description provides further details about the website analysed. Consequentially, the output depends on the tests that are applied to the examined website. A sample output of HoneyUnit for a set of 10 tests is displayed in Figure 5. During the respective analysis, four tests indicated suspicious content. The output can also be provided as a JSON document for processing by other tools or the CCH.

```
URI||XXXX
AnalysisTime||2464
TestTime||1011
NumTests||10
NumSuspicious||4
suspicious||testSuspiciousArguments(…): 1 very long string-argument(s) used on
an ActiveXObject
suspicious||testNopSlide(…): Possible NOP-Slide detected.
suspicious||testHeapSprayingArray(…):  Detected  possible  Heap-Spraying  array:
800 entries,  all very long Strings and equal
suspicious||testSuspiciousUnescape(…):  Method  unescape()  returned  suspicous
value. Possibly shellcode.
NumExploits||0
NumExceptions||0
DONE
```

Figure 5: Sample output of HoneyUnit

6.3. PDF Scrutinizer

Content delivered by malicious websites is not necessarily limited to HTML documents and JavaScript. As stated above, malicious websites can also deliver malicious PDF documents in order to attack a victim's browser or PDF rendering plugin respectively. To be able to analyse this content as well, PDF Scrutinizer dynamically analyses the content of PDF documents and identifies malicious patterns or behaviour. It can be used for example in conjunction with HoneyUnit to expand its detection capabilities allowing a more comprehensive and thus more accurate analysis of malicious websites.
In order to determine whether a PDF document is malicious, PDF Scrutinizer simulates the rendering of a PDF document within a commonly used PDF displaying application. During this process, the examined document is analysed for malicious structural content as well as for malicious behaviour of JavaScript in the document.
This is achieved by parsing the PDF document using PDFBox and extracting embedded JavaScript code. The extracted code is analysed by executing it through Mozilla Rhino and observing its runtime behaviour. To be able to observe this behaviour as detailed as possible, the API of common PDF rendering applications had to be emulated, allowing the JavaScript code to access content within the PDF document. During the emulation, libemu is used to detect potential shellcode used by the PDF document to exploit vulnerabilities. The general process of analysing PDF documents using PDF Scrutinizer is depicted in Figure 6.

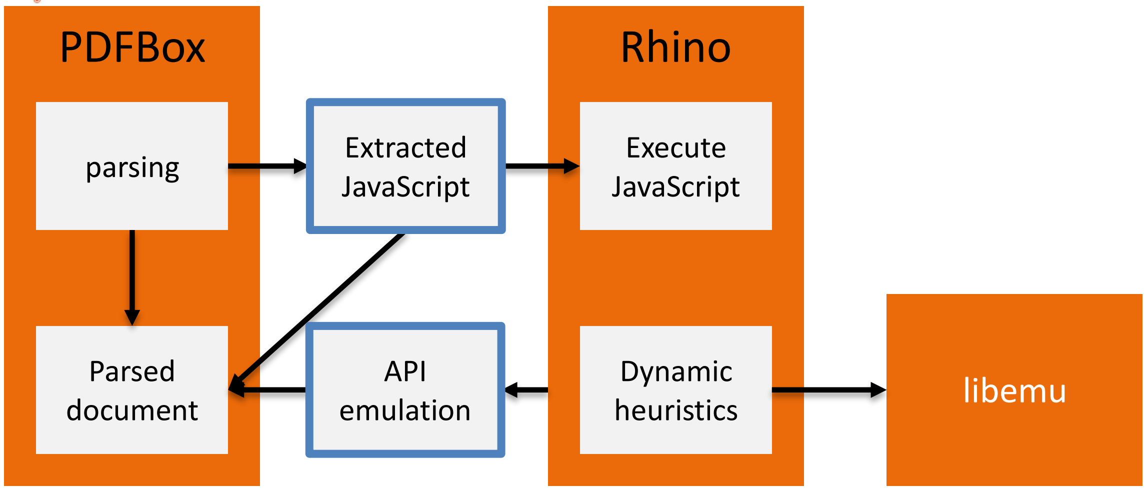

Figure 6: Overview about the functionality of PDF Scrutinizer

For detecting suspicious or malicious content, PDF scrutinizer features the following detection capabilities:

- StringLengthTester: PDF Scrutinizer detects unusually long strings, e.g. strings containing more than 100,000 characters, to identify potential NOP-sleds or shellcode.
- HeapSprayDetector: To detect heap spraying performed by a malicious JavaScript, PDF Scrutinizer detects large arrays containing sequences of similar data.
- ShellcodeTester: To detect shellcode within suspicious strings, libemu is used.
- Signatures: Regular expressions are applied to the original JavaScript code as well as dynamically generated code to identify known attacks. Signatures can be used to identify either suspicious or malicious content.
- VulnerableMethodCalls: PDF Scrutinizer employs a list of vulnerable methods to check whether a Script relies on potential vulnerabilities.

6.3.1. Input

PDF Scrutinizer is able to analyse local PDF files, or PDF documents provided by a webserver. Therefore, PDF Scrutinizer either requires a URL pointing to a PDF document online or the content of a PDF document to analyse.

To be able to detect current CVEs, PDF Scrutinizer also requires updated signatures. These signatures can be provided to PDF Scrutinizer in form of a regular expression, for example “`util.printf\(\s*\"%45000f\"`” for CVE-2008-2992.

6.3.2. Output

After completing its analysis, PDF Scrutinizer displays a list of raised heuristics as well as a list of CVEs that were detected through signatures. A sample output of PDF Scrutinizer is displayed in Figure 7. As shown, the analysed PDF document contained four malicious characteristics. Moreover, three known CVEs could be detected within the document. Consequentially, PDF Scrutinizer sets the final classification to “malicious”. To provide more detailed information about the attack, PDF Scrutinizer stores enclosed PDF documents or detected shellcode for further analysis. The output can also be provided as a JSON document which could be processed by other tools or the CCH.

```
Analysis start:          Fri Aug 23 12:38:36 CEST 2013
Analysis end:            Fri Aug 23 12:38:46 CEST 2013
Analysis time:           00:10
Filename:                11ae5cb8c47b59c2555dc8d680822de537d57539
MD5 Hash:                7048fb5e07d4f0aae1ee1b0d25f0e1ff
JavaScript events:       true
JavaScript count:        1
heuristics raised:       RegexMalicious ShellcodeTester HeapSprayDetector
StringLengthTester
vulnerabilities found:

CVE-2008-2992
-------------
API method: Util.printf
Type: buffer overflow

CVE-2007-5659
-------------
API method: Collab.collectEmailInfo
Type: buffer overflow

CVE-2009-0927
-------------
API method: Collab.getIcon
Type: buffer overflow

Classification:          malicious
```

Figure 7: Sample output for PDF Scrutinizer

6.4. SiteVet

SiteVet is a web service that provides data on malicious activity hosted worldwide. Data is combined from multiple sources – community partners as well as CyberDefcon's own research data – and processed using unique algorithms to provide meaningful results.
The focus is on Autonomous Systems and the "reputation" of hosts.

6.4.1. Requirements

SiteVet requires the provision of ASN data as well as descriptions for malware instances provided in the MAEC format and vulnerabilities in CVE and CVRF formats.

6.4.2. Output

Using the input data provided, SiteVet generates a reputation score for Autonomous Systems.

6.5. Skanna

Skanna is a tool for discovering websites using vulnerable software or serving known exploits. Currently, it receives lists of domain names from two third parties but also supports manually submitting domain names for analysis. When analysing a domain, the index page of a WWW server serving content under that domain is retrieved and scanned to detect the software and technologies employed using WhatWeb. The page will also be stored and indexed for later reference. In a second stage, the page will be scanned with an AV solution to identify known malicious content and possible compromisation. Results are currently stored in a local database and made available through a web interface.
To integrate Skanna into the ACDC solution, Skanna will be extended to analyse websites of interest, e.g. potentially malicious websites, and provide both data gathered and analysis results to the CCH.

6.5.1. Input

Skanna requires the submission of website addresses that should be analysed. The tool is currently designed to retrieve lists of domain names provided by third parties and manual submissions, but will be extended to allow the submission of more versatile representations of internet addresses, e.g. URLs.

6.5.2. Output

Analysis results are currently provided through a web interface and include (where applicable):

- Status of the website, i.e. whether it is online or not
- Software and technologies used by the scanned website
- Results of antivirus scan

Additional features such as full-text search and statistics on previously analysed websites are provided through that interface but are not specific to a given website's analysis.
To integrate Skanna with the ACDC solution, information on software and technologies as well as a brief version of the AV solution's scanning results will be provided to the CCH as data elements associated with a given website. While services such as full text search may not be appropriate for direct integration with the CCH, Skanna can provide indicators that provide information on how to obtain access to these services. Statistics derived from the data available to Skanna, providing a general overview on the scanning results, can be provided to the CCH both on a regular schedule and upon request.

6.6. WebCheck

WebCheck, provided by CyberDefcon, is a server plugin for webmasters that identifies, mitigates and remediates malware and vulnerabilities hosted from the server. It focuses both on cleaning websites and on forging trust by guaranteeing a website is safe.

6.6.1. Input

WebCheck relies on the provision of descriptions for malware instances provided in the MAEC format as well as vulnerabilities provided in CVE or CVRF formats. In addition, WebCheck requires the provision of attack patterns provided in CAPEC format.

6.6.2. Output

WebCheck will report data on instances of vulnerabilities discovered to the CCH.

7. Inter-Tool Communications

This section discusses the potential approaches for implementing inter-tool communication with respect to the Malicious or Vulnerable Website Tool Group and describes the solution deemed to be most appropriate with respect to that discussion.
We briefly introduce a few terms in section 7.1 and then continue to describe the implications of communications between tools within the tool group. Section 7.3 discusses communications with other ACDC components and describes the protocol for that interaction as a series of message exchanges.
In the following, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

7.1. Data Model

Throughout this section, we will be discussing data provided by different entities within the ACDC solution. Below, we will distinguish between data sets and data elements. A data element is a single atomic datum that reflects e.g. a measurement, a fact, the result of a tool's analysis or any other piece of information that should be provided to other entities.
Data elements may be grouped in data sets to indicate that they refer to one particular entity, e.g. a host, website or physical device, or incident. Each data set stored in the CCH MUST receive a unique ID to unambiguously identify the given data set throughout the ACDC solution. We will refer to this ID as data set ID or simply ID for short.

7.2. *Interaction within the Malicious or Vulnerable Website Analysis Tool Group*

Analysis of malicious or vulnerable websites is a complex task that requires information from various tools. In certain cases, however, the information provided by an individual tool may not be sufficient to reliably determine whether a website is vulnerable or malicious. In these cases, it could be necessary for the tools involved to interact with each other in terms of exchanging information. This interaction may be useful either if the output of a tool is required by another tool to start its analysis, or if one tool is used to augment the results of another tool. Both cases are discussed in the following as well as the consequences for the interaction between tools of tool group 1.1.4.

7.2.1. *Providing Data Triggering Independent Analysis*

Various tools of tool group 1.1.4 provide data or information that other tools may use for starting their own analysis. By allowing a direct interaction between those tools, other tools could start their analysis immediately after the first tool completes its analysis. For example, AHPS might detect an attack against a webserver and provide the respective URL to HoneyUnit or SiteVet for further analysis.

7.2.2. *Tools Augmenting Previous Analysis*

This case is similar to the previous case but has one major difference. If a tool is used to augment the results of another tool, its output needs to be transmitted back to the first tool. For example, HoneyUnit might detect a PDF document within an examined website and use the results of PDF Scrutinizer to enrich its own results. It is important to note here that PDF analysis results are not required by HoneyUnit and thus, this case cannot be translated into a cyclic version of case one.

7.2.3. *Communication aspects of tool group 1.1.4*

With regards to interactions between tools of tool group 1.1.4, the major concerns are how the design for an interface that covers the aforementioned cases should look like and which tool should transmit the final result to the CCH. Figure 8 illustrates the input data required by some tools as well as the data provided by the individual tools of tool group 1.1.4. Data that will be provided but is not used by any other tool is omitted for the sake of clarity. As the figure illustrates, some tools provide information that can be exploited by other tools. Currently, this includes URLs, domain information as well as malware listings. Since these will also be provided to the CCH, a specialised interface for intra-tool group communication would actually provide a subset of the functionality required for tool to CCH communication. Given the effort diverted from improving the ACDC core services for designing and implementing a separate protocol, intra-tool group communication should use the same protocol as tool to CCH communication.
Solving the second concern can be achieved by either determining a tool that should collect and store partial results until all contributing tools provided their share of the result, by creating a tool specifically for that task, i.e. collecting partial results and

submitting them to the CCH, or by allowing the submission of partial results that will be updated as new information becomes available. Each of the former two approaches would require the distribution and maintenance of a significant amount of configuration throughout the tool group. This particularly requires every tool to have up-to-date information about all its communication partners.

In the first approach, each tool involved in creating a result set would at least need to know which tool was responsible for submitting the respective results; for the second approach, the designated tool would have to know each expected result set. In either case, failure to update or inconsistencies across configurations for tools maintained by different partners could lead to either the unavailability or duplication of data sets at the CCH. Thus, data sets should be stored in the CCH immediately and tools should request a notification on changes, i.e. the addition of data to a data set that would allow them to provide additional analysis.

As pointed out in section 5.4 however, tools must not transmit large volume, unprocessed data to the CCH. If a tool provides large volume data to another tool, there is good reason to assume that the former acts as a sensor on behalf of the latter. Thus, these tools become components of a single logical tool and are thus free to define their own interface for intra-tool communication.

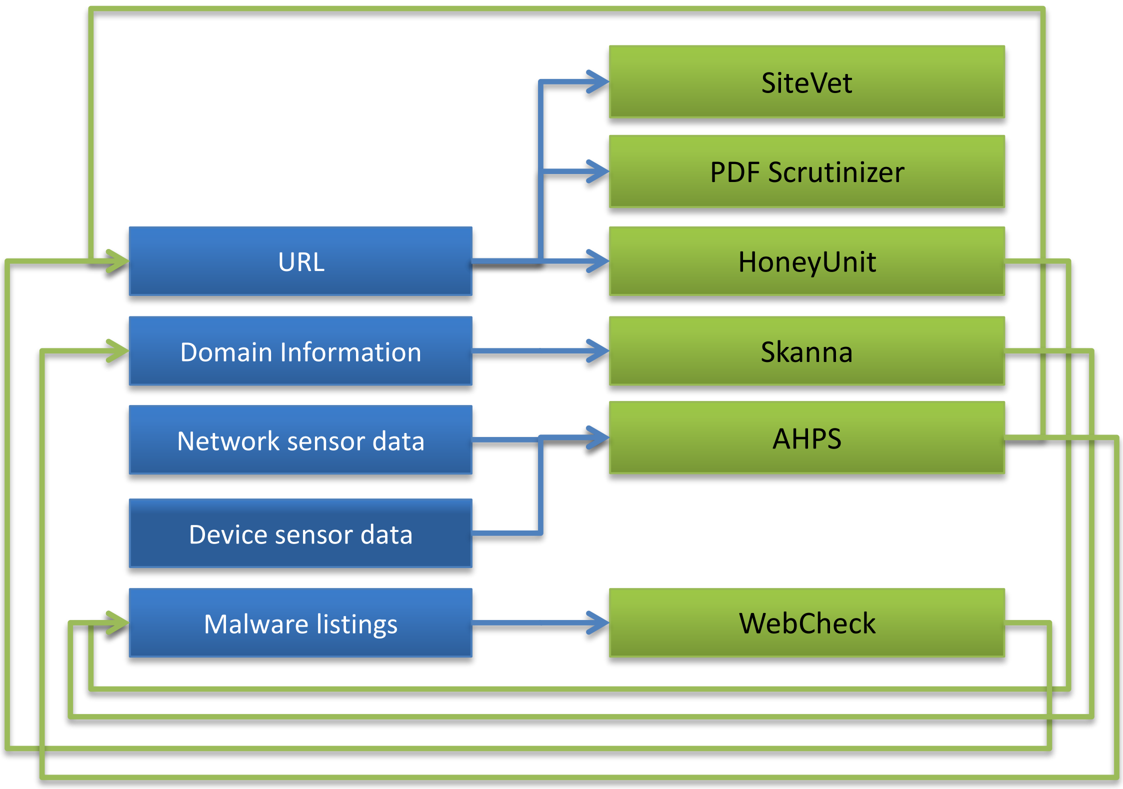

Figure 8: Information required and provided by tools of tool group 1.1.4. Generated information that is not required by other tools is omitted for the sake of clarity.

7.3. Interaction with other ACDC components

Integrating the tools in the Malicious and Vulnerable Website Tool Group with the ACDC solution requires them, though generally designed as standalone solutions, to interact with other components within the ACDC framework. In this section, we describe a design that allows for such interaction.

In section 7.3.1, we discuss different models for inter-tool communication with respect to how data sets should be disseminated throughout the solution. Thereafter, we describe in detail how tools should interact with the CCH. We argue that there should be no

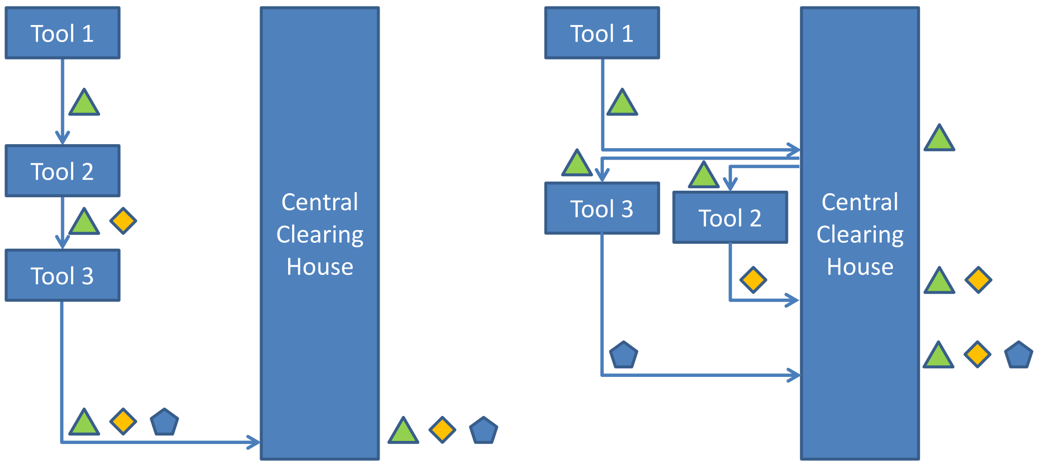

Figure 9: Visualisation of two approaches for organising data exchange within the ACDC solution. Geometric shapes represent data exchanged or stored.

decentralised tool-to-tool communication and close this section with a quick recap on the reasons in section 7.3.3.

7.3.1. Scope of Data Transmissions

To provide a benefit for ACDC stakeholders, data generated by the tools or information derived therefrom has to be stored in the CCH where it is used to assemble comprehensive reports for the stakeholders. Some tools may require data generated by others or be able to provide additional information when a certain datum becomes available. Thus, the respective data should be made available to both the CCH and those tools with as little delay as possible.

As the analysis in section 7.2 already suggests, there are several approaches for achieving this goal. The most simple approach would be to submit data sets from one tool to exactly one other tool, enriching each data set with the current tool's analysis results and submitting the whole data set to the CCH once all other tools provided their analysis. If any tool would be temporarily unavailable or require a lot of time for providing its analysis, all other data would be delayed as well.

This could be alleviated by instructing each tool to transmit its data to both the CCH and any tool that may rely on its results. Such a solution would however require the distribution of the input requirements throughout the ACDC solution. As tools and the input they need are likely to evolve, be added or removed over the lifetime of the solution, this would represent a significant challenge to the maintenance of the tools. In this scenario, changing the required input for a given tool would require the configuration for all tools providing its input to be adjusted as well, possibly triggering yet another round of configuration changes. Until the configuration of all tools affected was updated, the modified tool would not be able to provide its service. This equally applies if a single tool changes its output data, since all tools requiring this data may need to be adjusted. Similarly, adding or removing an entire tool would require all its communication partners to be adjusted. A more general challenge of tool-to-tool communication is that every tool has to be able to contact its communication partners. If these tools were deployed within different independent networks, this would require the tools to be reachable from the Internet.

To remove the need for distributing requirements throughout the solution, each tool could submit any data generated to all other tools in the solution. The amount of unnecessary data transmissions created by such an approach would however have the potential to cripple the whole solution. In addition to that, each tool would still need an up-to-date configuration for establishing a communication channel with each of the other tools. Thus, this approach is infeasible.

Considering the above approaches, the malicious and vulnerable website analysis tool group avoids messaging to other tools in favour of enriching data sets already stored in the CCH. This will not only decouple the tools, i.e. allow them to evolve without directly relying on specific changes being made to other tools or their configuration, but also ensures that any datum generated by any tool in the tool group will be available at the CCH as fast as possible. Therefore, the CCH would be able to provide preliminary reports in case a stakeholder is in urgent need of information on a particular threat, e.g. law enforcement or industry partners coordinating a takedown effort against a botnet.

Figure 9 illustrates a comparison between a decentralised pipeline approach discussed in the second paragraph on the on the left hand side and the favoured approach of exchanging data through the CCH. Geometric shapes represent data generated by tools, arrows represent message exchanges. Time progresses from top to bottom, i.e. a long vertical arrow indicates a long processing time. In the example depicted on the left hand side, tool 1 generates a piece of data that both tool 2 and 3 will use to generate additional data. While tool 2 can provide its results shortly after tool 1 supplied its data, no information is available at the CCH before the time consuming analysis of tool 3 is complete. Using the same requirements and processing times but relaying data through the CCH, as suggested above, results in a larger overall count of messages, but data becomes available at the CCH much earlier and, due to the implicit parallelisation of the processing by tools 2 and 3, even the overall duration of the processing is reduced significantly.

7.3.2. Communication with the Central Clearing House

As discussed in section 7.3.1, we argue that communication should only take place between tools and the CCH, i.e. there should be no direct tool-to-tool communication. In this section, we describe how tools should interact with the CCH, starting with the role of the entities in network layer communication. The next sections discuss security requirements and subscription management. Finally, we discuss the message exchanges required to implement this protocol in section 7.3.2.4.

7.3.2.1. Roles in Network Layer Communication

When a tool communicates with the CCH, each of the peers may assume one of three roles with regard to the OSI Layer 3/Network Layer:

- Client
- Client and server (peer-to-peer mode)
- Server

To establish a communication channel, a client has to be aware of the server's network address. While e.g. the Domain Name System simplifies obtaining the respective address, the client must have advance knowledge on each server it may want to communicate with, e.g. its domain name. If the CCH would assume a client role, it would require that this information is always available and up to date for each tool provided to the ACDC solution.

In a dynamic environment, where tools may change, possibly move from one physical machine to another, be split into several smaller or merged into one larger tool, each of these changes would need to be carried out with close involvement of the CCH maintainer or otherwise the respective parts of the solutions would be unavailable. This would however increase the cost for maintaining the CCH without providing a significant benefit to the ACDC solution. Thus, the CCH should not assume a client role with respect to the network layer.

The second option would be to use a peer-to-peer channel, i.e. both the CCH and the tool it would be communicating with would assume each role at times. A major drawback of this approach is that the CCH would assume a client role, which is not desired as discussed above. Moreover, this would require individual tools to assume a server role. This is also undesirable since it would require tools to be reachable from the Internet. Finally, implementing a peer-to-peer channel would also require a lot of configuration effort since each tool would have to maintain a list of communication partners. As a result, this would imply significantly increasing the overall cost for implementation and maintenance and thus we discourage the use of peer-to-peer for the website analysis to CCH communication channel.

Finally, tools could assume a client role on the network layer, contacting a CCH server. This approach resembles the structure of the ACDC solution with the CCH providing a central location for storing and refining data gathered by the individual tools. While tools will need to store the host name or network address of the CCH to be able to establish a communication channel, particularly a host name is likely to be long lived. Changes to a tool, including splits, merges and relocations, will remain transparent to the CCH since it does not require any details on the client to establish the communication channel. Thus, tools in the website analysis tool group will initiate connections on the network layer when they need a service provided by the CCH or are able to provide a service to the CCH.

7.3.2.2. Confidentiality, Integrity and Access Control

To ensure the confidentiality and integrity of data exchanged between website analysis tools and the CCH, all connections should be encrypted and their integrity be checked. In addition, access, in particular with regard to operations that add or modify data, must be limited to trusted parties. These properties will be ensured by the technology framework developed as part of Task 2.2.

A possible approach for achieving the goals stated above is to wrap any message exchange in Transport Layer Security (TLS/SSL) sessions, using client certificates issued by a trusted party for authenticating the client’s identity. The CCH should monitor the issuer’s Certificate Revocation List for changes and ensure that connections, including open connections, using revoked certificates will be closed or rejected.

7.3.2.3. Subscription Management

Providing all data sets to each tool would create an unreasonable amount of unneeded messaging, particularly since data sets would need to be retransmitted each time they were updated. In addition to that, legal or contractual limitations may exist regarding what data a given tool or its operator may access. Thus, a mechanism must be implemented that allows managing subscriptions, taking into consideration the aforementioned aspects. Preferably, such a mechanism should be self-serviceable for the tool developers to facilitate implementation of new or improved approaches and reduce the administrative overhead for the CCH operator. While further details of such a mechanism are out of the scope of this document, we assume that it provides tool operators with an API key or similar mechanism that will be used both for selecting the appropriate subscription as well as for providing proof that the tool is authorised to access that particular subscription.

Given such a mechanism, all a tool developer needs to do once its tool’s requirements change is to obtain an API key for the respective subscription. As soon as its tool establishes a new communication channel with the CCH, providing the new key, it is integrated with the ACDC solution.

Note that some tools may be able to provide different analysis depending on the data elements available. Therefore, a subscription may contain more than one set of data elements that would satisfy the requirements of a tool. To improve distinguishability between sets of data elements stored in the CCH and sets of data elements requested in a subscription, we will call the latter a “group of data elements” or “group” hereafter. Each of these groups must receive an ID which is unique with respect to the subscription they are associated with.
Whenever the CCH creates a new data set or on and only on the addition of data to an existing data set, it will evaluate or re-evaluate whether that data set now satisfies any subscriptions that had previously not been satisfied. Thus, each tool may be served each data set at most once for each group of data elements requested in a given subscription.

7.3.2.4. Message Exchanges

This section provides a list of message exchanges, indicating the intention and actions to be taken by each peer when sending or receiving such a message. Since the technical implementation of the messages should consider the format selected for representing data that will be part of deliverable 1.7.2, it cannot be defined in full detail at this time. Thus, the content of the respective messages, provided in the appendix, will only be defined on a high level using the EBNF syntax in this version of the document. Note that this allows for implementing all the logic required for these message exchanges using classes or similar abstractions that only leave the representation/parsing of the messages in transit open.
Throughout this section, we will make extensive use of the terms “group” or “group of data elements” as defined in section 7.3.2.3.

7.3.2.4.1. General Mode of Operation

Messages are standardised representations of an intent, data or the result of an operation. Within this solution, messages MUST NOT be interleaved, i.e. while a part of a message may be delayed for arbitrary reasons, the communication channel MUST NOT be used by the party sending that message for any other means than completing that message. There will be two types of message exchanges, namely notifications and request-response exchanges. For the latter, after receiving a request, the responding party MUST NOT send any message other than the response to the given request. If a message was sent but not completed while receiving the request, it SHALL however be completed before sending the response message.
Notifications consist of a single message only and require no further action by the recipient. At this time, notifications are sent by CCH only. Request-response exchanges on the other hand MUST only be initiated by a tool and consists of a message requesting an action by the CCH and a response indicating that the request was processed and possibly further details, where needed. The CCH SHOULD buffer an appropriate number of notifications generated while a message exchange is incomplete but MAY drop older notifications, if the count of unsent notification exceeds the count deemed appropriate by the CCH maintainer. Tools SHOULD avoid the loss of notifications by establishing separate communication channels for delivering data to and receiving data from the CCH.
The CCH MUST NOT accept requests from a client that has not been successfully authenticated. The CCH MAY process submissions if a client was authenticated successfully through other means but otherwise MUST NOT send notifications or process message exchanges other than the authentication/subscription request

exchange unless such an exchange has been completed successfully for a given communication channel. If an error occurs at any stage of receiving, parsing or processing a request, the CCH MUST close the communication channel to indicate that an error occurred. If this document defines a response message indicating a specific error condition, the CCH SHOULD however send that message instead of closing the communication channel.
Tool maintainers are free to choose an appropriate reaction to a failed message exchange, e.g. re-establishing the communication channel and repeating a request or silently ignoring an error message. The CCH MAY however impose a penalty, e.g. rejecting attempts to establish a communication channel for a short duration, if a tool appears to generate a significant amount of faulty requests. Tool maintainers SHOULD keep track of failed message exchanges to be able to discover errors or misconfigurations.

7.3.2.4.2. Authentication/Subscription Message Exchange

The Authentication/Subscription Message Exchange will be initiated by a tool after establishing a communication channel with the CCH. When no such message exchange has been completed successfully for a given channel yet, the CCH MUST NOT send any notification to the client and MUST reject any message exchange unless it is explicitly permitted to accept that message exchange by this document. Whenever a client initiates a Authentication/Subscription Message Exchange, the CCH MUST cancel any existing subscription for that client and treat it as if the communication channel had just been established.
If an error occurs when processing a request message, the CCH MUST close the respective communication channel. If a request was processed successfully, the CCH SHALL respond with a success message. Both closing the channel as well as sending a success message MAY be delayed by the CCH to mitigate brute force attacks, however, subscriptions MUST NOT become effective before a success message was sent to the client.

7.3.2.4.3. Submission Message Exchange

A Submission Message Exchange is initiated by a tool that wants to store data in the CCH. It MAY contain a data set ID assigned by the CCH to indicate that the data set associated with that ID should be updated by inserting the values supplied or by replacing them, if present. The details for representing that data will be defined in deliverables 1.2.1 and 1.7.2.
Once a submission is successfully parsed and stored in the CCH, it SHALL send a submission response success message. If parsing or storing the message fails, the CCH MUST close the communication channel to indicate that an error occurred.

7.3.2.4.4. Data Set Ready Notification

If after creating or modifying a data set that data set fulfils a subscription request that it did not fulfil before the given update, the subscriber associated with that subscription MUST be notified by the CCH. The notification SHALL include the data set ID for the updated data set and the ID of the group or groups of data elements that were satisfied by the data set for the first time.
There will be no response to the data set ready notification

7.3.2.4.5. Request Data Set Message Exchange

When a tool receives a data set ready notification and wants to process the respective data set, it will send a request for that data set. To indicate the data elements it needs, the request will include the IDs of the data set in the active subscription that should be satisfied by the data set. If only one group is associated with the given subscription, providing the ID of that group is optional.
Upon receiving a data set request, the CCH will search its database for a data set with the ID indicated in the request. If no data set with the given ID is available, it SHALL send a data set not available message to the tool, indicating the requested ID and completing the message exchange.
Otherwise, the CCH will calculate the join of all data elements requested in the groups referenced by the tool. Should any of those elements not be available, the CCH MUST complete the message exchange with a data set element not available message, indicating the requested data set's ID and the groups that could not be satisfied at this time.
Finally, if the requested data set is available and contains all requested data elements, it is encoded in accordance with the specification that will be provided in deliverable 1.7.2 and delivered to the tool to complete the message exchange.

7.3.3. Communication with other ACDC tools

The arguments given in sections 7.2.2, regarding data transmission within the tool group, and 7.3.1, regarding tool-to-tool communications in general, suggest that there is no immediate benefit of tool-to-tool communications. On the contrary, it comes at the cost of requiring the distribution and maintenance of configuration state regarding many, if not all, tools throughout the ACDC solution. This would increase the cost for operating the ACDC solution and divert resources from implementing or improving its components to protocol implementation and maintenance.
With the given lightweight protocol for tool to CCH communications, analysis is implicitly parallelised and results are provided to the stakeholders through the CCH as soon as possible. Thus, it is designated to remain the only protocol tools in the Malicious or Vulnerable Website Analysis Tool Group are required to implement.
We point out that the requirements defined in section 5 do however explicitly allow for tools to implement their own interface if they do need to exchange data with a given other tool. Generally, doing so should only be required for large volume data, implying that one tool would serve as a sensor to another tool, i.e. those tools would serve as a single logical tool from the ACDC solution's perspective.

8. Conclusion

Our analysis of the requirements for the very diverse set of tools contributed to the Malicious or Vulnerable Website Analysis Tool Group indicated that while technically possible, there is little benefit from implementing a communication protocol either within the tool group or for exchanging data with other tools directly. Storing data in the CCH immediately, on the other hand, provides benefits in several areas. First, it will be available to the stakeholders with as little delay as possible. Secondly, the approach decouples tools, simplifying and thus encouraging their extension as well as developing new tools or deploying existing tools at new sites. Finally, it reduces the effort required for implementing the protocol and cuts cost for maintenance, supporting a long term perspective for the ACDC solution.
This document thus defines how tools should interact with the CCH on a technical level. It also provides the argument for a self-serviced subscription model, allowing developers to define or redefine the data needed for their analysis while they are still working on their implementations.

While some details of the implementation should not be defined without knowing the data format the CCH will use for providing data sets to the tools in the solution, we provide a high level description of the respective messages that allow tool maintainers to implement the larger part of the respective interfaces. The exact representation of those messages will be provided in a later revision of this document.

9. Annex

9.1. Messages

This section describes the content of messages exchanged between the CCH and a tool using the EBNF syntax.

9.1.1. Authentication/Subscription Request Message

```
API Key = *Authentication and subscription identification key provided
by external subscription method*
Subscription Request Message = API Key
```

9.1.2. Subscription Response Success Message

```
Subscription Response Success Message = *Subscription Successful*
```

9.1.3. Submission Request Message

```
Data Set ID = *Unique Identifier for a data set, assigned by the CCH*
Tool Data = *Data generated by the initiating tool*
Submission Request Message = [Data Set ID,] Tool Data
```

9.1.4. Submission Response Success Message

```
Data Set ID = *Unique Identifier for a data set, assigned by the CCH*
Submission Response Success Message = Data Set ID
```

9.1.5. Data Set Ready Notification

```
Data Set ID = *Unique Identifier for a data set, assigned by the CCH*
Subscription Request ID = *Identifier for a request, chosen by the
sender of the respective subscription request message*
Nonempty List of Subscription Request IDs = Subscription Request ID[,
Nonempty List of Subscription Request IDs]
Data Set Read Notification Message = Data Set ID[, Nonempty List of
Subscription Request IDs]
```

9.1.6. Request Data Set Message

```
Data Set ID = *Unique Identifier for a data set, assigned by the CCH*
Subscription Group ID = *Identifier for a requested set of data
elements, associated with the currently active subscription*
Nonempty List of Group IDs = Subscription Group ID[, Nonempty List of
Group IDs]
Request Data Set Message = Data Set ID[, Nonempty List of Group IDs]
```

9.1.7. Requested Data Set Not Available Message

```
Data Set ID = *Unique Identifier for a data set, assigned by the CCH*
Requested Data Set Not Available Message = Data Set ID
```

9.1.8. Requested Data Set Element Not Available Message

```
Data Set ID = *Unique Identifier for a data set, assigned by the CCH*
Subscription Group ID = *Identifier for a requested set of data
elements, associated with the currently active subscription*
Nonempty List of Subscription Group IDs = Subscription Group ID[,
Nonempty List of Subscription Group IDs]
```

```
Requested Data Set Element Not Available Message = Data Set ID, Nonempty
List of Subscription Group IDs
```